ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hriadelová** Jméno: **Anna Mária** Osobní číslo: **466067**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vizualizace změn v procesních aplikacích

Název bakalářské práce anglicky:

Change visualization in process application

Pokyny pro vypracování:

Analyzujte systém verzování procesních aplikacích na platformě IBM BPM.
Navrhněte a implementujte nástroj pro porovnávání zdrojového kódu mezi dvěma verzemi aplikací.
Změny provedené mezi verzemi zobrazte textově i vizuálně.
Připravte testovací scénáře a proveďte testování.
Připravte uživatelskou dokumentaci.

Seznam doporučené literatury:

Dyer L., Flournoy H., Lehmann I, aj.: Scaling BPM Adoption: From Project to Program with IBM Business Process Manager, IBM Corporation, 2012
IBM Corporation: Building process applications [online], 2017. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSFTBX_8.5.7/com.ibm.wbpm.wle.editor.doc/topics/proccessapp_highlevel_roadmap.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Zoubek, Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Lukáš Zoubek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studentky

Bakalárska práca

České
vysoké
učení technické
v Praze

F3 Fakulta elektrotechnická
Katedra počítačů

Vizualizácia zmien v procesných aplikáciach

Anna Mária Hriadelová

Vedúci práce: Ing. Lukáš Zoubek
Odbor: Softwarové inženýrství a technológie
Máj 2019

Poďakovanie

Rada by som poďakovala vedúcemu práce Ing. Lukášovi Zoubkovi a kolegovi Bc. Tomášovi Malinkovičovi za všetky pripomienky, rady a čas, ktorý mi venovali počas písania bakalárskej práce.

Prehlásenie

Čestne vyhlasujem, že som predloženú prácu vypracovala samostatne a že som uviedla všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác. V Prahe, 24. mája 2019

Abstrakt

Bakalárska práca sa zaoberá vytvorením návrhu a implementácie nástroja pre porovnávanie dvoch verzií procesných aplikácií vytvorených v IBM Business Process Manager. Riešenie má pomôcť vývojárom BPM pri porovnávaní rôznych verzií ich aplikácie, pretože existujúce riešenie nieje vyhovujúce.

Kľúčové slová: procesná aplikácia, Business Process Manager, BPMN 2.0, porovnávanie verzií, diff

Vedúci práce: Ing. Lukáš Zoubek

Abstract

The bachelor thesis deals with the design and implementation of a tool for comparing two versions of process applications created in IBM Business Process Manager. The solution is supposed to help BPM developers compare different versions of their application because the existing solution is not satisfactory.

Keywords: process application, Business Process Manager, BPMN 2.0, version comparison, diff

Title translation: Visualization of changes in process applications

Obsah

Obrázky

Výpisky

Tabuľky

Kapitola 1
Úvod

Počas môjho pôsobenia v Centre znalostného managementu na Fakulte elektrotechnickej ČVUT v Prahe som narazila na problém pri verzovaní v procesných aplikáciach. Vývojari na platforme IBM BPM pri verzovaní svojej aplikácie majú obmedzenú možnosť porovnávať uskutočnené zmeny súčasneho stavu aplikácie s niektorým z predchádzajúcich verzií. Verzovanie aplikácií je dôležité pri vyvíjaní komplexných aplikácií, obvzlásť tých, na ktorých pracujú viacerí vývojari zároveň. Pri dosiahnutí významného miľníku, uskutočnení zmeny alebo testovaní je potrebné si aplikáciu uložiť v tomto novom stave, aby mal vývojár možnosť sa vrátiť k pôvodnej verzii alebo porovnať uskutočnené zmeny. Takýto problém rieši systém na správu verzií Git, kde vývojari kopírujú verzie svojej aplikácie a majú možnosť porovnávať uskutočnené zmeny medzi jendotlivými verziami pomocou funkcie *diff*, ktorej výstupom je vizualizácia týchto zmien a vývojar tak vidí, čo v aplikácií pribudlo a naopak.

V IBM BPM sa verzuje aplikácia pomocou *snapshots* - okopírovanie stavu všetkých artefaktov v procesnej aplikácii v časovom bode, kedy bol snapshot vytvorený. Snapshot nesie informáciu o tom, kto je jeho autorom a kedy bol vytvorený. Vývojár si môže vybrať cieľový snapshot a vrátiť sa do jeho stavu no možnosť si tento snapshot porovnať so súčasným stavom aplikácie je nevyhovujúci. Pri porovnávaní snapshots sa zobrazia jednotlivé artefakty, ktoré nesú informáciu o tom, kto ich vytvoril a modifikoval s príslušným dátumom a časom. Tieto dáta nenesú informáciu o tom ako sa artefakty zmenili, čo obmedzuje vývojára pri revízií aplikácie.

Mojím cieľom tejto bakalárskej práce je navrhnúť riešenie vyššie uvedeného problému a implementovať ho.

Kapitola 2

Cieľ práce

Cieľom tejto práce je návrh a vytvorenie nástroja na vizualizáciu zmien v procesných aplikáciach vytvorených na platforme IBM BPM z informácií, ktoré získam pri analýze spôsobu fungovania systému na správu verzií Git. Implementovaný nástroj má pomôcť vývojárom BPM pri hľadaní rozdielov rôznych verzii procesnej aplikácie. Porovnávanie na tejto platforme je možné, ale pre vývojárov nevyhovujúce, pretože konkrétne zmeny v procesoch niesú zobrazené. Vytvorené riešenie bude slúžiť ako funkčný prototyp spomínaného nástroja, ktorý by sa v budúcnosti mohol rozšíriť v komunite BPM vývojárov.

Kapitola 3

Metodika a štruktúra

Na dosiahnutie uvedených cieľov v predchádzajúcej kapitole je nutné najprv zanalyzovať ako funguje porovnávanie súborov v systéme pre správu verzií Git, popísať prostredie IBM BPM a spôsob ako prebieha verzovanie procesnej aplikácie a porovnávanie jednotlivých verzií. Nazbierané poznatky tvoria teoretickú časť tejto práce a na ich základe navrhnem riešenie pre vizualizáciu zmien IBM BPM. V návrhovej časti taktiež objasním prečo som zvolené riešenie vybrala a popíšem jeho výhody a nevýhody.

Z návrhovej časti vyplýva implementácia riešenia. Na overenie riešenia vytvorím testovaciu aplikáciu s výraznými zmenami medzi verziami tak, aby bol cieľ bakalárskej práce pokrytý testami. Na základe získaných výstupov z overenia zhrniem, do akej miery sa mi riešenie podarilo dotiahnúť a aké sú jeho slabé a silné stránky.

Na začiatku každej kapitoly je stručné zhrnutie problematiky popísanej v danej kapitole a nakonci zhrnutie dosiahnutých výsledkov.

Kapitola 4
Verzovanie

V tejto kapitole opisujem systém pre správu verzií, analyzujem existujúce softvéry spomedzi ktorých vyberiem najvhodnejší z nich a zaoberám sa jeho spôsobom fungovania. Opíšem výhody verzovacieho systému a prečo ho používať a nakoniec popíšem dôležitú časť systému - funkciu *diff* a algoritmy, na základe ktorých tento nástroj funguje. Táto kapitola je dôležitým podkladom pre návrh riešenia problému bakalárskej práce.

4.1 Systém pre správu verzií

Podľa článku [9] je to systém schopný zaznamenávať zmeny uskutočnené v súbore alebo skupine viacerých súborov v priebehu času takým spôsobom, aby nám umožňoval vrátiť sa v čase do stavu, ktorý vyžadujeme. Vďaka tomu je možné zobraziť presný stav sledovaných súborov kedykoľvek v minulosti a pokiaľ súčasné úpravy spôsobili nežiadúce chovanie, je možné sa vrátiť k staršej verzii. Na zistenie rozdielu medzi dvoma súbormi spravidla medzi súborom starším a súčasným sa používa príkaz *diff*.

4.2 Existujúce systémy pre správu verzií

V súčasnosti existuje mnoho systémov pre správu verzií. Podľa článku [10] je najpoužívanejší Git (67%), Subversion (28%) a menej používané sú Mercurial (1%), Bazaar (1%) a CVS (Concurrent Version System) (1%). Git je distribuovaný systém oproti druhému najpoužívanejšiemu verzovaciemu systému Subversion, ktorý je centralizovaný.

Podľa článku [11] od autora knihy Pro Git používať distribuovaný systém má veľa výhod. Jedným z nich je, že takmer každá operácia je vykonaná mimo dát na lokálnom disku, čo znamená že môže byť vykonaná offline - Subversion oproti tomu nemôže takéto operácie vykonávať bez prístupu k internetu. Ďalšou výhodou Gitu je, že každý kto pracuje na rovnakom projekte má úplnú zálohu dát a má schopnosť synchronizácie s ostatnými repozitármi - to znamená, že do projektu je možné pridať viacero vzdialených repozitárov.

Kvôli týmto výhodám analyzujem verzovací systém Git ako podklad pre riešenie cieľa mojej práce.

4.3 Prečo verzovať

V dôvodoch prečo verzovať sa budem zameriavať na skupinu ľudí, ktorí vyvíjajú zdrojový kód. Rozvoj softvéru môže byť veľmi riskantný bez používania verzovacieho systému kvôli stráte dôležitých dát a komplikuje tak vývoj v tímovej spolupráci. Podľa dokumentácie [12] by mal používať verzovací systém každý, kto potrebuje:

- uchovávavať históriu zmien každého súboru - to znamená každú zmenu vykonanú jendotlivými vývojármi v tíme v priebehu rokov. Zmeny zahŕňajú vytvorenie a vymazanie súborov, ako aj úpravy ich obsahu od autora,
- spravovať viacero verzií toho istého súboru uloženého pod rovnakým názvom,
- mať možnosť vrátiť jednotlivé súbory alebo celý projekt do predchádzajúceho stavu,
- vytvárať nové vetvy, čo umožňuje pracovať nezávisle od ostatných a zároveň poskytuje možnosť zlúčenia, čo umožňuje vývojárom overiť, že zmeny na každej vetve niesú v konflikte,
- byť schopný sledovať každú zmenu vykonanú v projekte a komentovať každú zmenu so správou popisujúcou jej účel a zámer,
- zistiť, kto naposledy upravil niečo, čo môže teraz spôsobovať problémy,
- rýchlo a ľahko obnoviť súbory, ktoré sa stratili alebo boli zmenené.

4.4 Git diff

Git *diff* príkaz podľa článku [13] z anglického slova *difference* nám umožňuje vidieť rozdielnosti medzi ľubovoľnými dvoma súbormi, commits alebo working trees, atď. Je to vlastne funkcia, ktorá berie dva vstupy a výstupom je zmena medzi nimi.

Na obrázku 4.2 je zobrazený výstup príkazu *diff*. Prvý riadok príkazu zobrazuje vstupné zdroje diffu - to je spravidla súčasná a staršia verzia porovnávaného súboru. Môžeme vidieť, že `a/diff_test.txt` a `b/diff_test.txt` boli predané ako vstupy. Druhý riadok zobrazuje interné metadáta Git. Čísla v tomto výstupe zodpovedajú identifikátorom hash verzie objektov Git, čo je pre nás nepotrebná informácia.

Tretí a štvrtý riadok priradí symboly ku každému vstupnému zdroju. V tomto prípade, zmeny z `a/diff_test.txt` sú označené -- a zmeny z `b/diff_test.txt` sú označené symbolom +++.

Zvyšné riadky výstupu sú tzv. *diff chunks* - konkrétne „krajce" kódu, ktoré boli zmenené. Riadok označený @@ -1 +1 @@ sa nazýva *chunk header* - hlavička krajca a jeho obsahom je zhrnutie uskutočnených zmien v súbore.

Diff zobrazuje iba časti súboru, ktoré boli zmenené. V tomto príklade máme iba jednu zmenu a môžeme vidieť, že každý zmenený riadok je predpísaný symbolom + alebo -, ktorý označuje, ako sa jednotlivé vstupy zmenili. Ako som už uviedla predtým, symbol - označuje zmeny z `a/diff_test.txt` a symbol + označuje zmeny z `b/diff_test.txt`.

```
diff --git a/diff_test.txt b/diff_test.txt
index 6b0c6cf..b37e70a 100644
--- a/diff_test.txt
+++ b/diff_test.txt
@@ -1 +1 @@
-this is a git diff test example
+this is a diff example
```

Obrázok 4.1: Výstup príkazu git diff, zdroj [1]

Zmenu jedného súboru môžeme popísať textovým súborom s koncovkou *.diff*, ktorý nám popisuje aké zmeny musíme uskutočniť, aby sme sa dostali z jednej verzie na druhú. Tento vygenerovaný súbor označujeme **patch** (záplata) a nesie v sebe informáciu, ktorá je výstupom obrázku 4.2.

Diff funguje na základe algoritmov, ktoré sú uvedené podrobnejšie v ďalších podkapitolách. Užívateľ má možnosť si vybrať jeden zo štyroch možných algoritmov na porovnávanie rozdielov a to:

- **Myers diff algorithm**, základný algoritmus na nájdenie rozdielností s minimálnou dobou spracovania.
- **Minimal**, ktorý strávi čas naviac aby sa ujistil, že aj najmenšia možná zmena je zaznamenaná.
- **Patience**, kladie viac dôraz na čitateľnosť záplaty ako na veľkosť obsahu a dobu spracovania.
- **Histogram**, rozširujúci patience algoritmus na podporu málo vyskytujúcich sa spoločných prvkov.

4.4.1 Najdlhšia spoločná subsekvencia

Keďže základom každého z vymenovaných algoritmov v predchádzajúcej časti je podľa článku [7] *LCS* algoritmus - longest common subsequence, je potrebné ho definovať.

Princíp

Majme dve sekvencie označené A a B a ich dĺžky označme M a N. Úlohou algoritmu je nájsť dĺžku najdlhšej možnej subsekvencie - sekvencie, ktorá sa

vyskytuje v rovnakom relatívnom poradí ale nemusí byť nutne súvislá. V tomto prípade ide o najdlhší možný zoznam spoločných znakov v sekvenciách A a B.

Popis kódu

```
LCS(A,B,m,n)
    L[m,n] <- new table
    for i <- 0 to n
        L[i,0] <- 0
    for j <- to m
        L[0,j] <- 0
    for i <- 1 to n
        for <- 1 to m
            if A[i] = B[i] then
                L[i,j] <- L[i-1,j-1] + 1
            else
                L[i,j] <- max{L[i-1,j], L[i,j-1]}
return L[i,j]
```

Výpis 4.1: Najdlhšia spoločná subsekvencia (LCS) [7]

Funkcia LCS obsahuje 4 vstupné parametre - pole znakov sekvencie A a sekvencie B, dĺžku sekvencie A a sekvencie B a na výstupe vracia veľkosť najdlhšej subsekvencie. Na začiatku si inicializujeme dvojrozmerné pole L s danou veľkosťou.V prvých dvoch for cykloch určíme hodnotu LCS 0 ak je jeden zo znakov sekvencie A alebo B na nultej pozícii. Tretím for cyklom prechádzame prvé pole znakov, pričom jeden znak sa vo štvrtom for cykle porovnáva so znakom z druhej sekvencie do ich dĺžky a uchováva si hodnotu LCS. Ak sa posledné znaky zhodujú, dĺžka LCS je hodnota v predchádzajúcom riadku a stĺpci + hodnota 1 inak, dĺžka je vyššia hodnota LCS predchádzajúceho stĺpca a súčasného riadka alebo súčasného stĺpca a predchádzajúceho riadka.

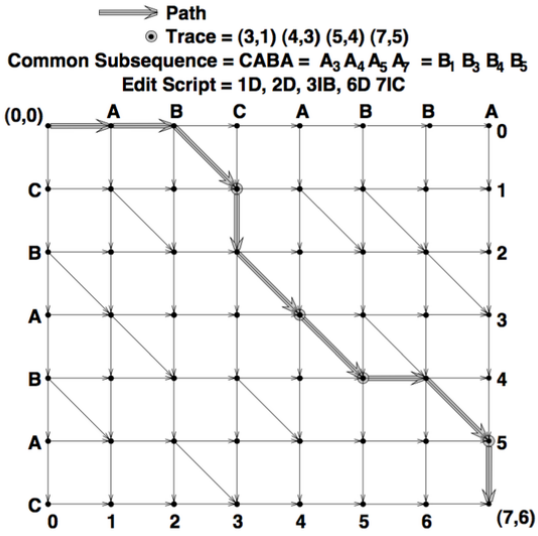
4.5 Myerov algoritmus

Abstrakt článku [2] uvádza problém hľadania najdlhšej spoločnej subsekvencie pri pretváraní sekvencie A do sekvencie B. Ekvivalentom tohto problému je nájdenie *shortest edit script* - najkratšieho skriptu na úpravu, ktorý hľadá minimálny skript obsahujúci symboly vkládania a mazania znakov pretvárajúci jednu sekvenciu do druhej.

4.5.1 Princíp

Myerov algortimus je založený na myšlienke, že nájdenie najdlhšej spoločnej subsekvencie, alebo ekvivalentné nájdenie najkratšieho skriptu na úpravu môže byť modelované ako vyhľadávanie grafov. Nech A a B sú naše sekvencie. Algoritmus zoberie vstupné sekvencie a vytvorí graf všetkých spôsobov, ktorými sa môžeme dostať z A do B. Pohyb doprava zodpovedá vymazaniu

znaku z A a pohyb smerom nadol zodpovedá vloženiu znaku z B. V niektorých pozíciách sa môžeme tiež pohybovať diagonálne. K tomu dochádza, keď majú dva reťazce rovnaké znaky na indexoch polohy, napríklad na obrázku 4.3 tretí znak v A a prvý znak v B sú oba rovnaké, takže nedošlo k mazaniu ani vkládaniu. Hlavnou myšlienkou algoritmu je prejsť graf z počiatočného bodu do koncového s čo najmenej krokmi, pričom chceme minimalizovať pohyby doprava a dole a maximalizovať diagonálny pohyb ako je na o obrázku 4.3 zobrazené.



Obrázok 4.2: Priechod grafom - Myerov algoritmus, zdroj [2]

4.5.2 Popis kódu

```
myers_diff(A,B)
    M <- A.length
    N <- B.length
    MAX <- M+N
    V <- Array[-MAX..MAX]
    V[1] <- 0

    for d <- 0 to MAX
        for k <- -d to d in steps of 2
            if k = -d or k != d and V[k-1] < V[k+1] then
                x <- V[k+1]
            else
                x <- V[k-1] + 1
            y <- x - k
            while x < N and y < M and A[x+1] = B[x+1] do (x,y)
                <- (x+1, y+1)
            V[k] <- x
            if x >= N and y >= M then
                return lenght of an SES is d
                stop
```

Výpis 4.2: Myerov diff algoritmus [2]

Algoritmus podľa článku [2] berie na vstupe počet riadkov súboru A a súboru B. Určíme si dĺžku vložených sekvencií, zadefinujeme pole V a začneme prechádzať graf od začiatočného bodu až po koncový. Ak sa nachádza na ľavom okraji, to znamená (k == -d) alebo ak sa nenachádza úplne hore (k! = d), algoritmus pokračuje smerom dole ak sa dostane na územie, ktoré ešte nebolo preskúmané pričom si uchováva históriu týchto krokov. Pokiaľ sa pohybuje diagnálne (while cyklus) znamená to, že sa znaky na rovnakej pozícií zhodujú. Ak algoritmus došiel do konca, vráti históriu krokov priechodom grafu a naopak ak nie, pokračuje na ďalšiu pozíciu.

4.6 Patience algoritmus

Patience algoritmus podľa článku [8] narozdiel od ostatných diff algoritmov kladie dôraz na málo vyskytované ale obsahovo veľké časti, ktoré slúžia ako ukazovatele dôležitého obsahu v texte.

Zlé sú prípady tie, v ktorých sa dve verzie veľmi líšia. Za týchto okolností môže byť algoritmus diff nesprávne zarovnaný tým, že sa zhoduje s dlhými úsekmi tzv.„curly" zátvoriek, ale vo vnútri môže brať zátvorky jednej funkcie v staršej verzii so zátvorkami novšej funkcie v súčasnej verzii.

Patience algoritmus je pomenovaný podľa patience sort algoritmu (radiaci algoritmus).

4.6.1 Princíp

Patience algoritmus je stále založený na LCS ale s podstatným rozdielom a to tým, že do úvahy berie len časti, ktoré označí ako ukazovatele.

Algoritmus označí prvé riadky v oboch súboroch a ak sa zhodujú, tak označí druhé, tretie, až kým sa riadky nebudú zhodovať. V druhom kroku označí posledné riadky, predposledné, atď. až kým sa nelíšia podobne ako v prvom kroku. Nájde všetky riadky, ktoré sa vyskytujú presne jeden krát na oboch stranách a prevedie LCS. V poslednom kroku opakuje prvý a druhý krok v každej časti zhodujúcej sa riadkami.

4.6.2 Popis kódu

```
patienceSort(seq)
    piles <- new array
    for each element el in seq
        for i <- 0 to piles.length
            if piles[i,-1] > el then
                insert el into piles[i]
                stop
            else
                insert [el] into piles
    return piles.length
```

Výpis 4.3: Patience radiaci algoritmus [8]

Výpis kódu 4.3 zobrazuje patience sort algoritmus. Na vstupe berie sekvenciu čísiel a na výstupe vracia číslo medzi 0 a dĺžkou sekvencie. Funkcia na začiatku zadefinuje tzv. kôpku, do ktorej sa vkládajú prvky zo sekvencie. Algoritmus prechádza každý prvok v sekvencii a porovnáva ho s prvkom z kôpky. Hodnota piles[i,-1] zaručí, že vrch kôpky je zoradený vzostupne. Ak existuje kôpka, ktorá má na vrchu prvok s vyšším číslom, prvok sa vloží do tejto kôpky. Ak takáto kôpka neexistuje, vytvorí sa nová a do nej sa vloží daný prvok.

4.7 Histogram

Histogram podľa článku [14] bol vytvorený ako rozšírenie *patience* algoritmu na podporu spoločných prvkov s malým výskytom. Výstupom tohto algoritmu je histogram toho, čo sa zmenilo.

4.7.1 Princíp

Základ algoritmu je vytvoriť histogram vyskytnutí sa každého prvku zo sekvencie A. Každy prvok zo sekvencie B je potom braný do úvahy striedavo. Ak prvok existuje aj v sekvencií A, a má menší počet výskytov, pozície sa berú ako kandidát na LCS. Po prejdení sekvencii B, LCS, ktoré má najmenší počet výskytov je vybrané ako rozdeľovací bod. Oblasť je rozdelená okolo LCS a algoritmus je rekurzívne aplikovaný na časti pred a po LCS.

Tým, že vždy vyberieme pozíciu LCS s najnižším počtom výskytov, tento algoritmus sa správa presne tak ako *patience* algoritmus keď medzi dvoma sekvenciami existuje jedinečný spoločný prvok. Ak neexistujú žiadne jedinečné prvky, namiesto nich sa vyberie prvok s najmenším výskytom.

4.8 Zhrnutie kapitoly

Neoddeliteľnou súčasťou vyvíjania softvéru je neustále pridávanie funkcionality. Tieto funkcionality majú kladný aj záporný dopad na fungovanie samotného softvéru a preto je dôležité si prácu verzovať. Na verzovanie práce sa používajú systémy pre správu verzií. Spomedzi dostupných verzovacích systémov som vybrala systém Git kvôli tomu, že je štatisticky najpoužívanejší a má najviac výhod. Vývojár, ktorý si verzuje prácu pomocou tohto systému má možnosť vidieť, čo do projektu pridal a vie sa jednoducho vrátiť k pôvodnej verzii. Výhody používania verzovacieho systému spočívajú hlavne pri vývoji v tíme, kde dochádza k neustálemu prepisovaniu súborov od rôznych užívateľov a tieto zmeny sa vo verzovaciom systéme automaticky spoja do jedného celku.

Dôležitou časťou Gitu je funkcia diff, ktorá dokáže porovnávať uskutočnené zmeny medzi rôznymi verziami projektu. Git využíva štyri algoritmy na vizualizáciu zmien a to Myerov (základný) algoritmus, ktorý je najpoužívanejší a zároveň najrýchlejší, minimal algoritmus, ktorý je odvodený od Myerovho

algoritmu ale časovo náročnejší, pretože jeho cieľom je zaistiť aby aj najmenšia možná zmena bola zaznamenaná, patience algoritmus kladie dôraz na málo vyskytované ale obsahovo veľké časti a histogram, ktorý bol vytvorený ako rozšírenie patience algoritmu na podporu málo vyskytujúcich sa spoločných prvkov a jeho výstupom je samotný histrogram zmien.

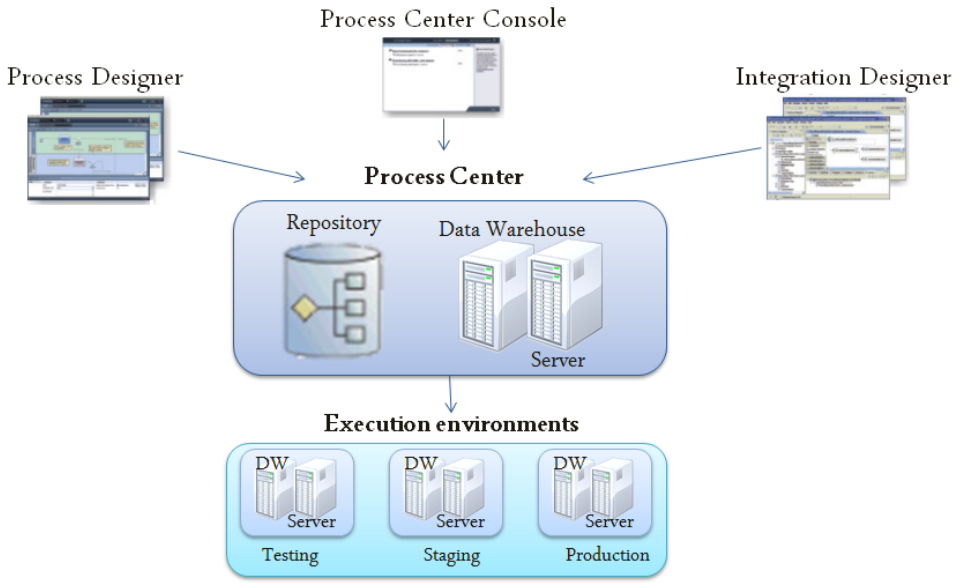
Jeden z uvedených algoritmov vyberiem ako podklad pre riešenie mojej práce.

Kapitola 5
Verzovanie v IBM BPM

V tejto kapitole budem analyzovať spôsob verzovania a definujem dôležité pojmy v IBM BPM.

5.1 IBM Business Process Manager

IBM Business Process Manager podľa publikácie [15] je komplexná platforma na riadenie business procesov. Poskytuje sadu nástrojov na tvorbu, testovanie a nasadzovanie business procesov, ako aj prehľad o ich riadení. IBM Business Process Manager je navrhnutý tak, aby uľahčoval prácu jednotlivých operácií a zabezpečuje úspešnejšiu spoluprácu so zákazníkmi. Na obrázku 5.1 je zobrazená architektúra IBM BPM. Najdôležitejšie komponenty IBM BPM sú detailnejšie popísané v nasledujúciach podkapitolách.



Obrázok 5.1: Architektúra IBM BPM, zdroj [3]

5.2 IBM Business Process Designer

IBM Process Designer podľa publikácie [15] umožňuje modelovať a implementovať business procesy podľa notácie BPMN (sekcia 5.6), namodelovať návrh a funkčnosť procesu počas vývoja aplikácie. Process Designer je pôvodone desktopová aplikácia založená na technológií Ecplise ale dostupná je aj novšia webová aplikácia.

Viacerí vývojári môžu súčasne pracovať a vytvárať nové položky v Process Designeri. Užívateľov, ktorí súčasne pracujú na rovnakom projekte je vidno na pracovnej lište a je možné sledovať ich aktivitu.

5.3 IBM Process Center

IBM Process Center podľa publikácie [15] slúži ako centrálny úložný priestor pre všetky časti projektu, ktoré sú vytvorené v Process Designeri ako aj samostatné procesné aplikácie. Užívatelia Process Designeru pripojení v Process Center majú možnosť zdielať medzi sebou rôzne procesy a služby. Taktiež môžu sledovať prebiehajúce zmeny v aplikácií v reálnom čase.

Process Center obsahuje Process Center Server a Performance Data Warehouse. Tieto funkcie umožňujú vývojárom na platforme Process Designer spúšťať procesné aplikácie a ukladať testovacie údaje počas vývoja bez toho, aby museli nasadiť aplikáciu na samostatný runtime server.

Z Process Center konzoly spravujú administrátori inštancie procesných aplikácií v nakonfigurovaných prostrediach.

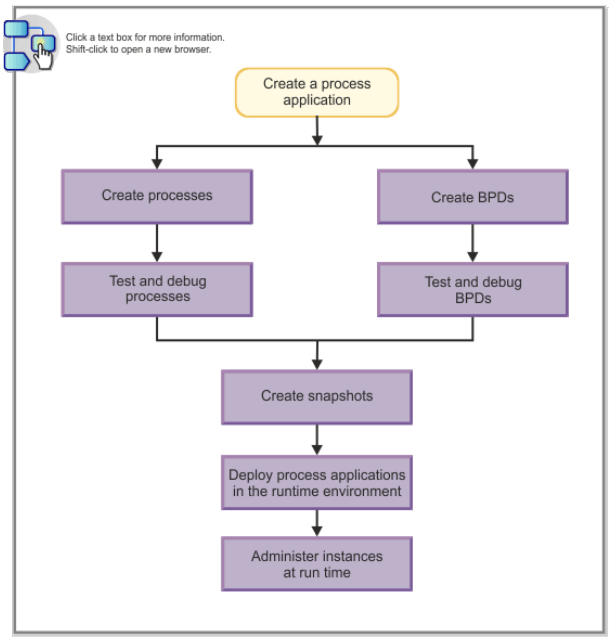
5.4 Procesné aplikácie

Podľa článku [15], v IBM BPM sú aplikácie vyvíjané ako procesné aplikácie BPM vývojarmi. Procesná aplikácia je kontajnerom procesných modelov a ich implementácií. Procesný model je grafické znázornenie jednotlivých aktivít alebo krokov procesu za sebou, kto ich vykonáva a čo je pri vykonávaní konkretného procesu potrebné. Každá aplikácia môže obsahovať niektoré z nasledujúcich procesných artefaktov v IBM Process Designeri (sekcia 5.2):

- **Business Process Definition (BPD)** - slúži na modelovanie procesu, ktorý je opakovane použiteľný.
- **General System Serivce** - slúži na koordinvoanej iných vnorených služieb alebo na manipuláciu s premennými údajmi.
- **Integration Serivce** - slúži na integrovanie externého systému.
- **Human Service** - užívateľská, interaktívna služba implementovaná V BPD.
- **Coach** - užívateľské rozhranie pre Human Services.

- **Business Object** - opisuje objekt a jeho atribúty.
- **UCA** - volá sa udalosťou na spustenie špecifickej služby
- **Team** - slúži na definovanie skupiny užívateľov, ktorí sú oprávnení vykonávať akcie týkajúce sa procesov a úloh.

Na obrázku 5.2 je diagram, ktorý zobrazuje základné kroky a aktivity spojené s tvorením procesnej aplikácie. Základnou aktivitou je vyvtorenie business process modelov a ich otestovanie, vytvorenie snapshots, nasadenie aplikácie a monitorovanie procesov administrátorom.



Obrázok 5.2: Diagram tvorenia procesnej aplikácie, zdroj [4]

Procesná aplikácia sa vytvaráva v Process Centre (sekcia 5.3) alebo je možné ju do procesného centra importovať. Aplikácia sa otvára v IBM Process Designeri, kde sa môžu vytvárať a upravovať business procesy.

Aplikácia a jej artefakty sú uložene v Process Center Repository - v repozitári procesného centra.

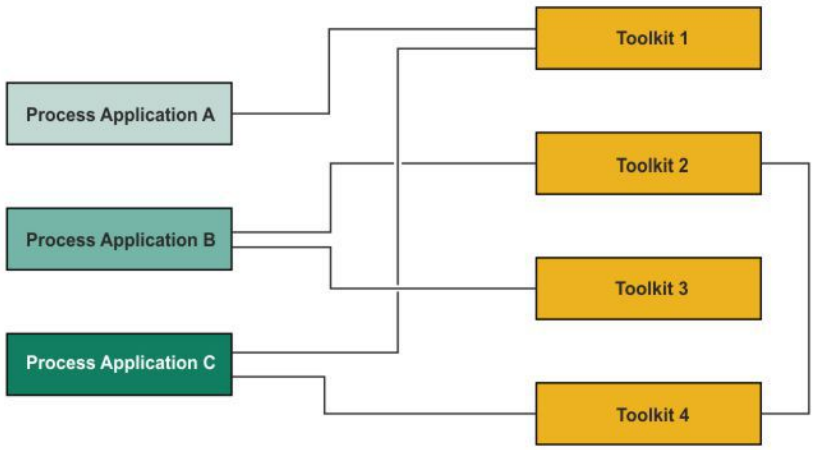
5.5 Toolkit

BPM toolkity podľa dokumentácie [5] alebo tzv. sady nástrojov sú knižnice artefaktov Process Designeru, ktoré obsahujú znovupoužiteľné business objekty, integračné služby alebo business procesy. Toolkity môžu byť zdieľané vo viacerých procesných aplikáciach ako je na obrázku 5.3 zobrazené.

Používatelia, ktorí majú prístup k toolkitu a chcú využívať jeho knižnice a položky, musia vytvoriť závislosť na daný toolkit. Pokiaľ sa toolkit aktualizuje

na novšiu verziu, v procesnej aplikácií je nutné zmeniť závislosť na novšiu verziu toolkitu.

Toolkity narozdiel od procesných aplikácií niesú spustiteľné a slúžia ako doplnok k nim.



Obrázok 5.3: Diagram prepojenia toolkitov a procesných aplikácií, zdroj [5]

5.6 Notácie BPMN

BPMN podľa článku [16] je grafická notácia, ktorá zobrazuje postupnosť krokov, ktoré na seba nadviazujú v business procese. Notácia bola navrhnutá tak, aby koordinovala postupnosť procesov a správ, ktoré prebiehajú medzi rôznymi účastníkmi procesu v súvisiacich aktivitách a má byť zrozumiteľná pre všetkých účastníkov procesu. Informácie v nasledujúcich podkapitolách sú čerpané z článku [6].

5.6.1 Swimlane

Plavecké dráhy (obrázok 5.4 - 1.5) slúžia k organizovaniu a kategorizovaniu aktivít a v BPMN sa skladajú z dvoch typov:

- **Pool** predstavuje hlavných účastníkov procesu, ktorý zvyčajne oddeľuje rôzne organizácie. Bazén obsahuje jeden alebo viac dráh.
- **Lane** sa používa na organizovanie aktivít v bazéne podľa funkcie a taktiež obsahuje správy, objekty a artefakty.

5.6.2 Activity

Aktivity sú základným kameňom každého procesu (obrázok 5.4 - 1.1). Je to činnosť, ktorú firma vykonáva. Jednotlivé aktivity sa môžu opakovať alebo môžu byť vykonávané paralelne.

5.6.3 Gateway

Brány umožňujú vetvenie a zlučovanie tokov alebo procesov (obrázok 5.4 - 1.4) a dajú sa rozdeliť na ďalšie typy:

- **Exclusive gateway** vytvára niekoľko ciest toku procesu ale tok procesu môže prebehnúť iba jednou z nich.
- **Inclusive gateway** sa používa v tom prípade, ak tok procesu môže prejsť cez bránu viac ako jednou cestou. Po prejdení brány sa väčšinou všetky toky zlúčia do jednej.
- **Parallel gateway** sa používa v prípadoch, keď tok procesu prechádza viacerými cestami naraz.
- **Event-Based gateway** je založená na udalosti, tok procesu pokračuje cez tú udalosť, ktorá nastene skôr.

5.6.4 Event

Udalosť predstavuje dej, ktorý má priamy vplyv na chod procesu. Ikony v kruhu označujú typ udalosti (napr. obálka predstavuje správu alebo hodiny reprezentujú čas). Udalosti sa tiež klasifikujú ako *catching*, napríklad prijatie správy spustí proces, a *throwing* napríklad odoslanie správy o dokončení ukončí proces. Udalosti delíme na:

- **Start event** (obrázok 5.4 - 1.2), ktorá predstavuje spúštač procesu, môže byť len typu catch a má prázdny okraj.
- **Intermediate event** predstavuje niečo, čo sa deje medzi začiatkom a koncom udalostí, je označená dvojitým okrajom.
- **End event** (obrázok 5.4 - 1.3) predstavuje výsledok aktivity alebo procesu, môže byť iba throw a je označená hrubým vyplneným okrajom.

5.6.5 Flow

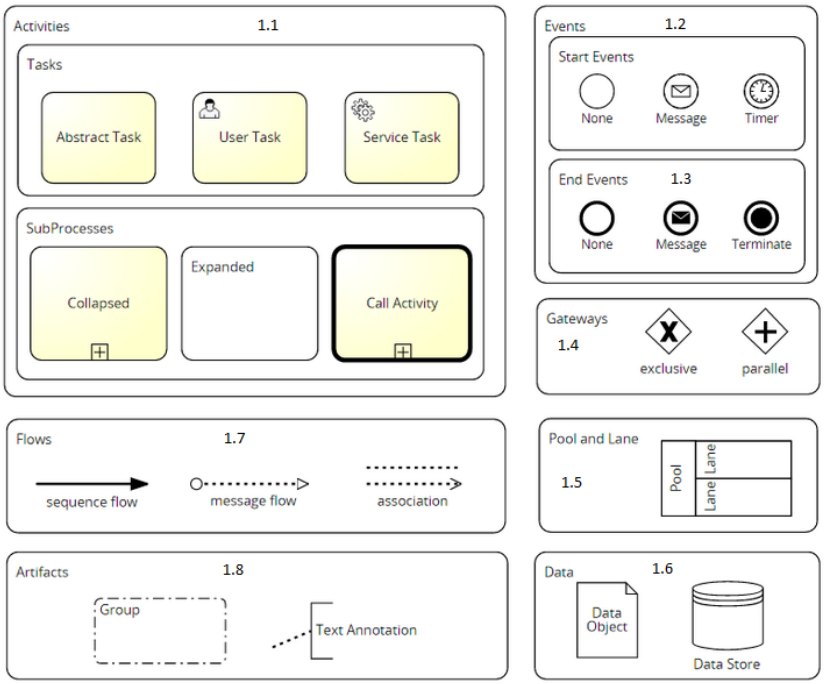
Správy alebo spojovacie objekty (obrázok 5.4 - 1.7) spojujú vyššie uvedené objekty a existujú 3 typy:

- **Sequence flow** reprezentuje v akom poradí sa vykonávajú činnosti.
- **Message flow** znázorňuje komunikáciu medzi účastníkmi procesu, prebiehajú naprieč bazénmi.
- **Association** sa používa na priradenie artefaktu alebo textu k objektom toku a môže určovať aj smer, ktorý označuje vstup alebo výstup.

5.6.6 Artifact

Artefakty umožňujú vývojárom pridať ďalšie informácie do modelu, aby tak bol proces čitateľnejší a jednoduchší na porozumenie. Existujú tri artefakty a sú to:

- **Data objects** (obrázok 5.4 - 1.6) zobrazujú čitateľovi, ktoré údaje sú vyžadované pri vykonávaní danej aktivity.
- **Groups** (obrázok 5.4 - 1.8) sa používajú na zoskupenie rôznych aktivít ale nemájú vplyv na tok v procese.
- **Annotation** (obrázok 5.4 - 1.8) slúži na opis častí diagramu kvôli lepšej zrozumiteľnosti.



Obrázok 5.4: Business Process Model and Notation, zdroj [6]

5.7 Verzovanie v IBM BPM

5.7.1 Snapshot

Snapshot podľa dokumentácie [15] označuje snímku procesnej aplikácie, zaznamenáva stav artefaktov v rámci aplikácie v konkrétnom čase a predstavuje konkrétnu verziu procesnej aplikácie. Snapshot obsahuje všetky komponenty, ktoré sú súčasťou procesnej aplikácie.

Snapshot je možné vytvoriť v Process Centre alebo v Process Designeri, ktorý je na Process Centrum napojený. Snapshots sa spravujú z konzoly

Process Centra, kde je možné ich kopírovať alebo medzi sebou porovnávať. Na nasadenie na samostatný procesný server, je potrebné vytvoriť takýto snapshot procesnej aplikácie, pretože tento artefakt je jediný, ktorý môže byť nasadený na procesné servery.

Snapshot je možné exportovať z Process Centra s príponou .twx - formát IBM, ktorý obsahuje informácie o snapshote a verzii, a môže byť opätovne použitý produktmi IBM BPM. Artefakty v tomto súbore sú popísane vo formáte XML a obsahujú všetky informácie o danom artefakte ako dátum vytvorenia a zmeny, autor, vstupné a výstupné dáta, atď.

Twx súbor obsahuje tieto adresáre:

- **META-INF** obsahuje údaje o exporte a package.xml, ktorý popisuje všetky artefakty v danom snapshote.
- **objects** obsahuje XML reprezentáciu všetkých artefaktov v aplikácii.
- **toolkits** obsahuje zip súbory všetkých závislých toolkitov na aplikácii.
- **files** obsahuje spravované artefakty.

5.7.2 Verzovanie

Podľa článku [17] verzovanie poskytuje možnosť identifikovať snapshots v cykle procesnej aplikácie a má mať schopnosť súčasne spustiť viacero snapshots na procesnom serveri.

Ako som v definícií procesnej aplikácie uviedla, procesná aplikácia je kontajner, ktorý obsahuje rôzne artefakty použité v aplikácií. Verzovanie sa vykonáva na tejto úrovni kontajnera a nie na úrovni jednotlivých artefaktov. V procesných aplikáciach verzovanie nastáva pri vytváraní snapshotu.

K týmto verziám sa vývojar môže kedykoľvek vrátiť a pretvoriť projekt späť do staršej verzie alebo môže na základe toho vidieť ako vyzerali jednotlivé artefakty v tejto verzii.

Každý snapshot má byť jedinečne pomenovaný podľa určitých dohodnutých konvencií aby bolo jednoznačné, v akom stave sa aplikácia nachádza.

Toolkity (sekcia 5.5), na ktorých je daná aplikácia závislá sa musia verzovať vždy potom ako v nich boli vykonané zmeny, ak ich chceme v aplikácií používať. V procesnej aplikácií je potom nutné aktualizovať novú verziu toolkitu.

5.7.3 Porovnávanie verzií

Vytvorené snapshots je možné medzi sebou porovnávať a zistiť, aké zmeny nastali vo verziách aplikácie. Pri dosiahnutí významného miľníku alebo ak došlo k veľkej zmene v projekte, vývojári si tieto zmeny verzujú. Jednotlivé verzie sú zobrazené v Process Designeri alebo v Process Centre aplikácie. Kliknutím na snapshot sa zobrazí jeho detail, v ktorom sú informácie kedy a kým bol snapshot vytvorený, nastavenia aplikácie, zoznam služieb a business objektov a kedy a kým boli naposledy zmenené. Tento snapshot je môžné

porovnať s iným snapshotom na zistenie vykonaných zmien. Porovnávanie medzi snapshots ukazuje aké artefakty boli vytvorené, vymazané alebo zmenené a obsahuje údaje o čase kedy zmena nastala a meno autora, ktorý túto zmenu spôsobil.

5.8 Zhrnutie kapitoly

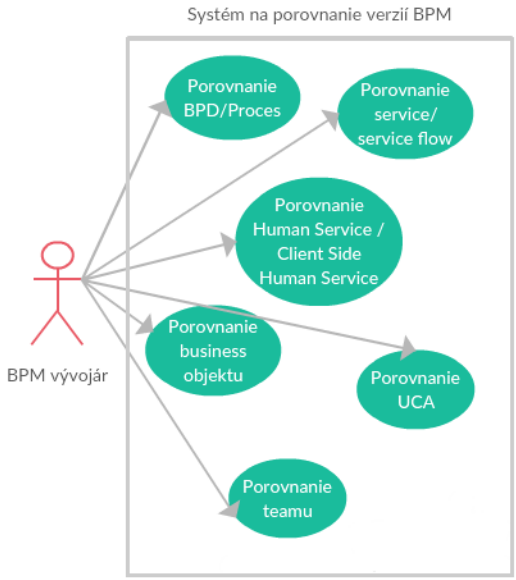
IBM BPM je platforma na riadenie business procesov, ktorá poskytuje sadu nástrojov na tvorbu, testovanie a nasadenie business procesov. Procesy sú modelované podľa štandardu notácie BPMN 2.0.

Verzovanie v IBM BPM prebieha pomocou vytváraní snapshots teda okopírovania stavu artefaktov procesnej aplikácie v danom čase. K jednotlivým snapshots má každý vývojar pracujúci na tej istej procesnej aplikácií prístup v Process Centre ako aj v samotnom Process Designeri. Snapshot je možné exportovať ako twx súbor, ktorý obsahuje celú business aplikáciu. Ak si chce vývojar pozrieť zmeny uskutočnené medzi dvoma verziami aplikácie, zobrazia sa mu informácie o tom, v akom artefakte nastala zmena spolu s príslušným časom uskutočnenej zmeny a meno autora, ktorý ju spôsobil. Tieto informácie ale niesú dostačujúce pre vývojárov ak chcú vidieť aká konkrétna zmena bola prevedená.

Kapitola 6

Návrh riešenia

V tejto kapitole sa zaoberám návrhom riešenia problému bakalárskej práce. Návrh vychádza z prípadov užitia (angl. Use Cases) ktoré sú zobrazené na obrázku 6.1.



Obrázok 6.1: Prípady užitia systému na porovnávanie verzií v IBM BPM

Prípady užitia môžeme rozdeliť do dvoch skupín a to porovnávanie založené na grafických zmenách v procesoch a porovnávanie na komponentoch, ktoré sú textovo popísané.

V prípadoch užitia porovnávania procesov (BPD proces, General System Service, Human Service) užívateľ zaznamenáva zmeny v grafických komponentoch, ktoré popisuje notácia BPMN 2.0. Zmeny užívateľ vidí graficky s identifikátormi zmien pri vymazaní, pridaní alebo zmenení artefaktov. Užívateľ tiež vidí, čím je proces definovaný (parametre, účastníci) a taktiež parametre samotných aktivít, ktoré sa v procese nachádzajú.

V prípadoch užitia porovnávania textových komponent (UCA, Business objekt, Team) užívateľ zaznamenáva zmeny v týchto artefaktoch ako textové

zmeny, ktoré popisujú tieto komponenty.

6.1 Porovnávanie zmien procesov

Najdôležitejším požiadavkom na porovnávanie zmien v procesných aplikáciach je samotná grafická reprezentácia zmien v procesoch. Vykonávané kroky v procese sú popísané ako udalosti, brány alebo aktivity, ktoré môžu obsahovať ďalšie služby či systémové, užívateľské alebo integračné potrebné na ich vykonávanie, ako som uviedla v podkapitole 5.4. Tieto procesy sú popísané v XML súboroch, ktoré obsahujú informácie o tom, z akých častí sa proces skladá a každý komponent má detailne definované súradnice výskytu, z akých komponent vychádza a na ktoré ďalšie komponenty nadväzuje. Taktiež sú na komponente popísané dáta, ktoré doň vchádzajú (angl. input parameters) a vychádzajú (angl. output parameters). Typ komponentu môže byť aktivita (užívateľská, systémová, integračná aktivita alebo script), udalosť (začínajúca, končiaca a intermediate) a brána (inclusive, exclusive, event-based) a podľa typu komponentu sa líšia aj informácie, ktoré obsahuje.

Pri grafickom porovnání procesov je nutné XML súbory parsovať (rozobrať/analyzovať) aby sme získali informácie, ktoré potrebujeme na porovnávanie a následné vykresľovanie procesu a zaznamávania informácií ako o procese, tak o každom jeho komponente. Procesy sa vykreslia ako grafy, v ktorých vrcholy reprezentujú komponenty a hrany reprezentujú sekvencie spájajúce tieto komponenty. Každý vrchol grafu nesie informáciu o jeho vlastnostiach, tak ako v procese.

Zmeny v procesoch majú byť viditeľné farebne, aby bolo užívateľovi jasné, aké zmeny nastali v oboch súboroch.

6.2 Porovnávanie textových zmien

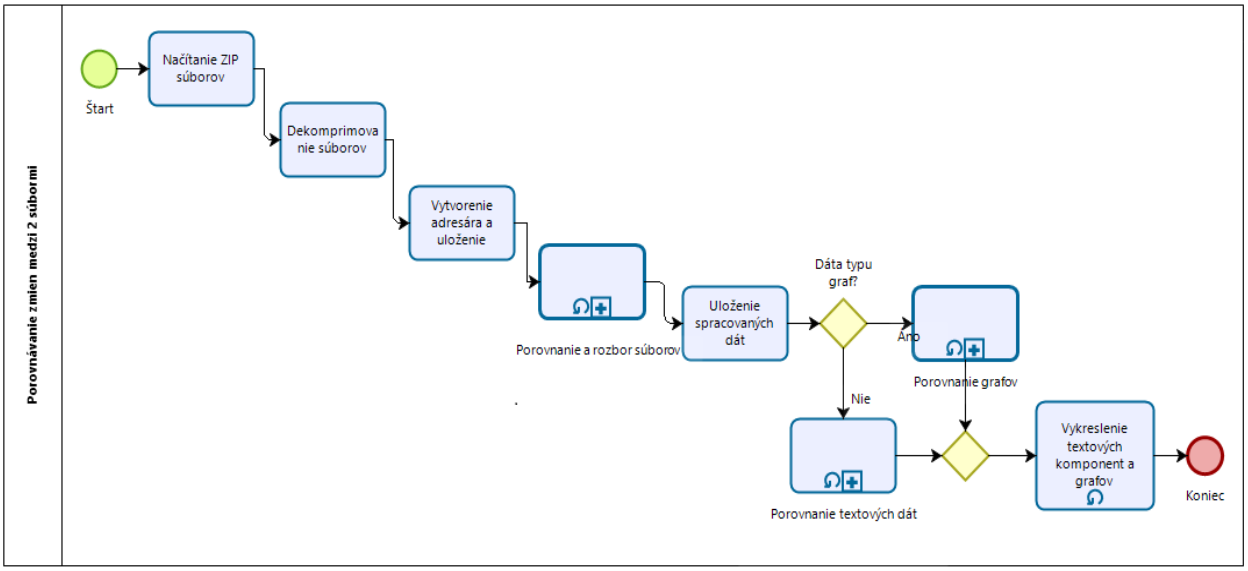
Ďalším veľmi dôležitým požiadavkom je zaznamenávanie zmien v Business objektoch, UCA a Teamoch. Tieto artefakty sú použité v samotných BPD procesoch a službách. Business objekt tvorí klasický objekt s atribútmi a dátovou štruktúrou. Team popisuje skupinu účastníkov podieľajúcich sa na procese alebo aktivite a UCA je služba, ktorá je volaná určitou udalosťou na spracovanie požadovanej služby. Tieto artefakty sú taktiež popísané v XML súboroch, takže je zase potrebné získať údaje, ktoré chceme porovnávať.

6.3 Popis systému

Proces porovnávania medzi verziami popisuje diagram aktivít na obrázku 6.2. Proces sa začína načítaním vybraných komprimovaných súborov, ktoré užívateľ vybral ako vstupné argumenty pre aplikáciu. Následne sa súbory dekomprimujú a uložia do vytvoreného adresára. V ďalšom kroku sa každý súbor porovnáva s identickým súborom v druhej verzii, kde zisťuje veľkosť súboru.

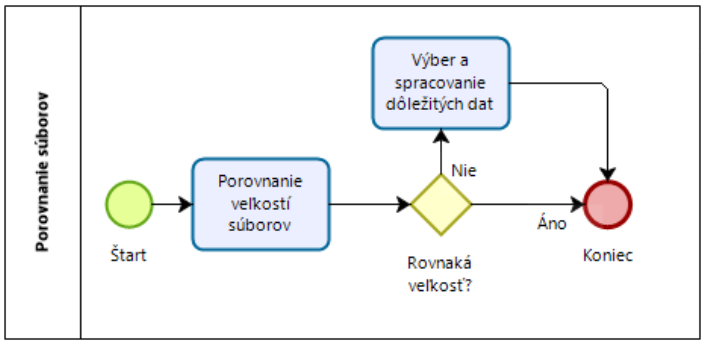
Pokiaľ sú uložené dáta typu graf - čiže proces, porovnajú sa grafy. Rovnako ako pri grafoch, porovnávame dáta uložené typu text.

V poslednom kroku všetky porovnané dáta vykreslíme do grafického rozhrania a tým sa proces ukončí.



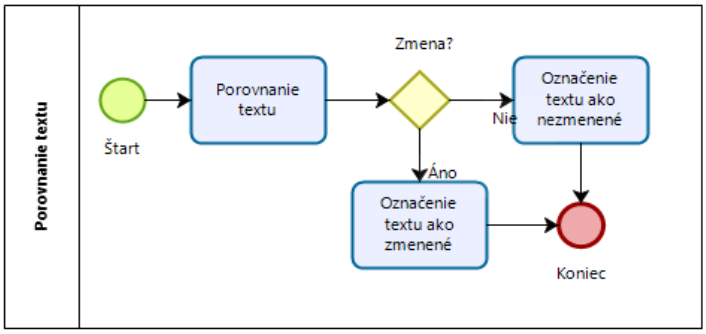
Obrázok 6.2: Diagram aktivit popisujúci proces porovnávania zmien

Podproces na obrázku 6.3 popisuje porovnávanie veľkosti súboru, na základe ktorej sa rozhodne, či bol súbor zmenený alebo nie. Proces je popísaný ako slučka, kým sa neporovná každý jeden súbor v oboch verziách. Súbory líšiace sa vo veľkosti sú parsované a následne uložené podľa toho, či sa jedná o proces alebo artfakt popísaný textovo.



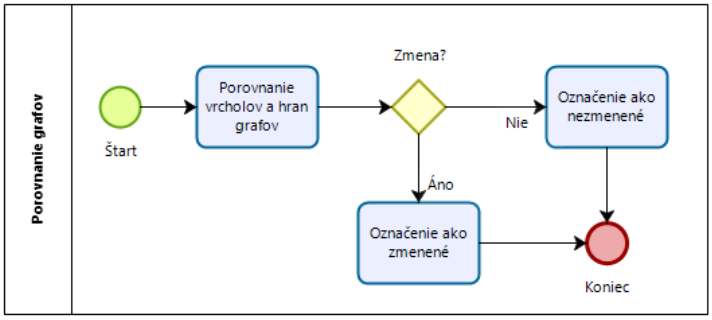
Obrázok 6.3: Proces porovnávania súborov

Porovnávanie textu prebieha v podprocese na obrázku 6.4, kde porovnávame string so stringom a pokiaľ sa hodnota zmenila, označíme textovú komponentu ako zmenenú a zmenu si uložíme.



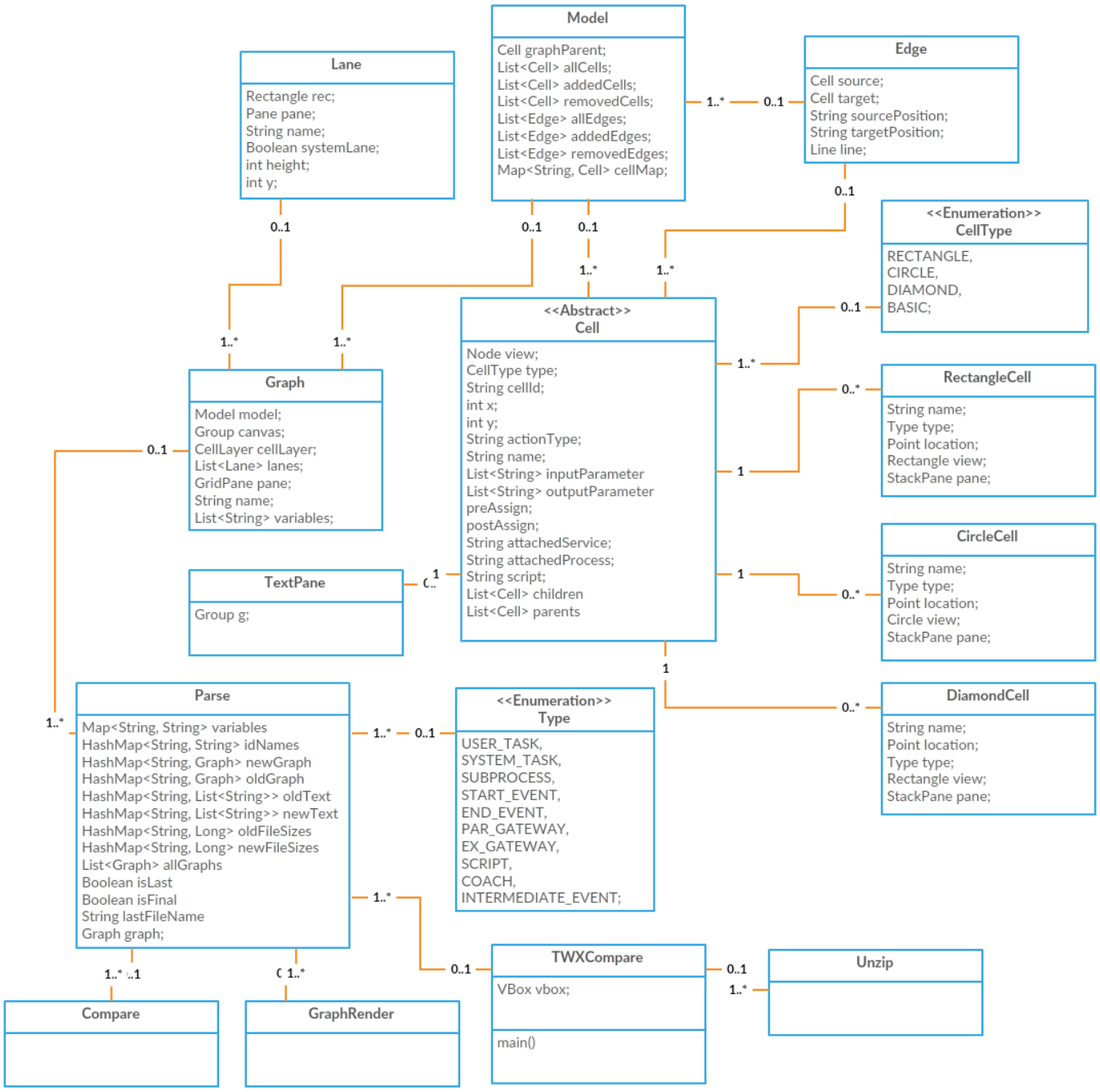
Obrázok 6.4: Proces porovnávania textových dát

Podproces na obrázku 6.5 porovnáva grafy, kde sa porovnajú všetky vrcholy a hrany dvoch grafov. V každom vrchole sa porovnávajú aj jeho parametre aby sme zistili, či nedošla zmena v nich. Pokiaľ zmena nastala, označíme vrchol alebo hranu za zmenený.



Obrázok 6.5: Proces porovnávania grafov

Návrhový model tried (angl. Class Diagram) je popísaný na obrázku 6.6, ktorý znázorňuje triedy systému a jeho atribúty.



Obrázok 6.6: Návrhový model tried

Jednotlivé triedy sú popísané v tabuľke 6.1.

Trieda	Popis
TWXCompare	trieda, v ktorej sa nachádza spúštiaca metoda
Unzip	slúži na dekomprimovanie ZIP archívu
Parse	rozhoduje, či sa súbory zmenili a parsuje ich
Compare	porovnáva zmenené artefakty
GraphRender	vykresľuje grafické znázornenie zmien
Type	popisuje typ komponenty
Graph	reprezentuje triedu graf
Model	vytvára vrcholy a pripisuje im hrany
Cell	reprezentuje vrchol grafu
CellType	popisuje typ vrcholu
RectangleCell	vrchol typu obdĺžnik
CircleCell	vrchol typu kruh
DiamondCell	vrchol typu kosoštvorec
Edge	reprezentuje hranu v grafe
Lane	popisuje swimlane v procese
TextPane	reprezentuje textové komponenty

Tabuľka 6.1: Popis tried návrhového modelu

6.4 Zhrnutie kapitoly

Návrh riešenia je založený na požiadavkoch prípadov užitia spomínaných v tejto kapitole. Na vytvorenie návrhu bolo potrebné zistiť, ako sú opísané artefakty v XML súboroch po exportovaní verzie procesnej aplikácie. Každý artefakt je potrebné parsovať, aby sme získali všetky potrebné informácie, ktoré môžeme potom medzi sebou porovnávať. Artefakty popisujúce daný proces (BPD, služby) obsahujú všetky komponenty, ktoré sa v ňom nachádzajú a pri každom komponente dokážeme získať informácie o jeho identifikátore, type, pozícii, názvu, parametroch či podmienkach. Tieto údaje je potrebné uložiť, aby s nimi program mohol ďalej pracovať pri porovnávaní a nakoniec grafickom vykresľovaní.

Druhý typ artefaktov sú textové komponenty ako Business objekty, Teamy - čiže účastníci procesu a UCA služby. Tieto typy artefaktov sú na porovnávanie jednoduchšie, pretože sa medzi nimi porovnáva iba text.

Diagram aktivit popisuje aké kroky má program vykonať, aby sme získali požadovaný výstup a návrhový model popisuje akú štruktúru by aplikácia mala mať.

Po konzultáciach s vedúcim práce sme zatiaľ nenašli využitie na diff algoritmus porovnávajúci verzie ako na gite, pretože tento algoritmus nedokáže správne porovnávať XML súbory a tak by sme nedokázali získať všetky informácie, ktoré sú potrebné na vykresľovanie procesov. Jediné racionálne využitie tohto algoritmu je pri porovnávaní zdrojového kódu v komponentoch typu script.

Kapitola 7

Implementácia

Táto kapitola popisuje implementáciu systému pre porovnávanie verzií procesných aplikácii. Implementácia je rozdelená na 4 hlavné časti ako bolo popísané na obrázku 6.2 v diagrame aktivít. Aplikácia je naprogramovaná v jazyku Java s využitím JavaFX ako platformy na vytváranie užívateľského rozhrania.

7.1 Vstupné argumenty, dekompresia a prehľadávanie súborov

Aplikácia berie ako vstup pri spúšťaní dva argumenty, a to cesty k cieľovým súborom, ktoré chce užívateľ porovnávať. Zadané súbory musia byť komprimované ZIP archívom. Program tieto argumenty odovzdá objektu na dekomprimovanie súborov a uloží ich.

Po dekomprimovaní súborov sa cyklom získavajú údaje o ich veľkosti, ktoré sa ukladajú do HashMapy - mapa, kde sa uchováva unikátny kľúč s príslušnou hodnotou. Kľúče sú v tomto prípade názvy súborov. Každý súbor je unikátne pomenovaný a jeho hodnota je jeho veľkosť. Po získaní údajov o všetkých súboroch, ktoré chceme porovnávať, t.j. súbory s príponou .xml získame dve HashMapy. Jedna mapa obsahuje údaje o súboroch v staršej verzii, druhá o súboroch v novšej verzii.

Po získaní veľkostí program začne cyklom prechádzať najprv staršiu verziu, kde potrebujeme získať údaje zo súboru „package.xml", ktorá obsahuje všetky identifikátory a názvy artefaktov v aplikácií. Tieto údaje si znova uložíme do HashMapy s kľúčom identifikátora a hodnotu s názvom artefaktu. Keďže niektoré artefakty majú referenciu na identifikátory iných artefaktov, takto si ľahko získame ich názov, ktorý požadujeme ako výstupnú hodnotu referencie.

Následne hľadáme súbory v adresári „objects", ktorý obsahuje všetky artefakty uložené v XML súboroch. Tu využijeme HashMapu s údajom o názve a veľkosti súboru. Ak sa názov súboru zhoduje s názvom v druhej HashMape, porovnáme ich veľkosti. Ak sa veľkosť nezmenila, znamená to, že v súbore nenastala žiadna zmena a tak pre nás nie je dôležitý, preto prechádzame cyklom na ďalší súbor. Pokiaľ sa ale veľkosti nezhodujú, potrebujeme získať všetky dôležité údaje o tomto artefakte. Tieto informácie získame tzv. parsovaním - čiže „rozoberaním" súboru. Ak sa daný názov súboru nezhoduje so žiadnym

názvom v druhej HashMape znamená to, že bol vymazaný a na výstupe bude označený ako „Deleted file: <názov artefaktu>".

Tieto kroky opakujeme pri prechádzaní súborov novšej verzii aplikácie, avšak ak sa daný artefakt v staršej verzii nenachádza, označíme ho ako „New File: <názov artefaktu>".

7.2 Parsovanie súborov

Ak chcem získať informácie z XML súborov musíme ich parsovať. Java používa na parsovanie XML súborov objekty, ktoré potrebujeme importovať do triedy, ktorá ich využíva.

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

Výpis 7.1: XML-súvisiace balíky objektov

Ako prvý potrebujeme vytvoriť DocumentBuilder, ktorý definuje rozhranie na získanie inštancií DOM z dokumentu XML. Vytvorenie DocumentBuilderu popisuje nasledujúci výpis:

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
doc.getDocumentElement().normalize();
```

Výpis 7.2: Vytvorenie DocumentBuilderu

Každý XML súbor obsahuje elementy (prvky), ktoré môžu mať atribúty a každý element môže byť popísaný inými elementami atp. V procesných aplikáciach rozoznávame šesť základných elementov - process (klientské, integračné, systémové služby), BPD (business process definition), underCoverAgent (UCA), twClass (business objekt), participant (účastník procesu) a coachView (užívateľské obrazovky). Každý z týchto elementov má ďalšie iné elementy, ktoré popisujú daný artefakt.

Program získa informáciu o tom, aký základný element popisuje daný artefakt a podľa toho tento súbor zpracuje. Príkladom získania elementu je nasludujúci výpis:

```
1 doc.getElementsByTagName("process")
```

Výpis 7.3: Získanie elementu

Tento element si zapíšeme do NodeListu - rozhranie poskytujúce usporiadanú kolekciu uzlov (angl. nodes) bez toho, aby definovala alebo obmedzovala spôsob implementácie tejto kolekcie. NodeList potom prechádzame cyklusom a získavame ďalšie elementy, ich hodnoty, ktoré sú pre nás podstatné. Nasledujúci výpis popisuje reprezentáciu XML súboru, ktorý chceme parsovať:

```
<process id="1.0fb51fa2-6b92-4940-b6aa-18853783bfd1" name="Send
    refuse mail (Recruit)">
    <processParameter name="email">
```

```
        <parameterType>1</parameterType>
        <isArrayOf>false</isArrayOf>
        <classId>1a8bbfd8-919c-4d0e-8ce4-15162936522a/
        12.db884a3c-c533-44b7-bb2d-47bec8ad4022</classId>
    </processParameter>
</process>
```

Výpis 7.4: XML súbor reprezentujúci process

Vidíme, že element process má atribúty id a name, ktoré potrebujeme získať ako jednu z informácií pre porovnávanie artefaktov. Element process začneme prechádzať cyklusom a získavame element, v ktorom máme informácie o parametroch procesu. Tento element je znovu typu NodeList, pretože vidíme, že obsahuje ďalšie elementy. Uložíme si názov parametru, ktorý je jeho atribút a získavame informácie o tom, či ide o vstupný alebo výstupný parameter. Pokiaľ ide o vstupný parameter je označený hodnotou 1 a výstupný hodnotou 2. Element <isArrayOf> nám dáva informáciu o tom, či sa jedná o pole a preto je popísaný hodnotami true a false. Element <classId> obsahuje referenciu na typ atribútu, ktorý môže byť základný (String, Integer atp.) alebo je to business objekt, ktorý sme si v rámci aplikácie definovali. Túto referenciu potom vyhľadáme v HashMape, v ktorej máme uložené všetky objekty a pokiaľ sa tam nachádza, získame jeho hodnotu ako typ atribútu. Pokiaľ nie, prechádzame HashMapu, v ktorej sú uložené všetky typy základných atribútov, ktorú sme získali z toolkitov.

Nasledujúci výpis popisuje parsovanie predchádzajúceho výpisu 7.4:

```
NodeList nList = doc.getElementsByTagName("process");
for (int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    Element e = (Element) nNode;
    String id = e.getAttribute("id");
    String processName = e.getAttribute("name");

    NodeList nL1 = e.getElementsByTagName("processParameter");
    for (int i = 0; i < nL1.getLength(); i++) {
        Node nNode1 = nL1.item(i);
        Element e1 = (Element) nNode1;
        String eName =  e1.getAttribute("name");
        String type =
            e1.getElementsByTagName("parameterType").item(0).getTextContent();
        String classId =
            e1.getElementsByTagName("classId").item(0).getTextContent();
        String array =
            e1.getElementsByTagName("isArrayOf").item(0).getTextContent();
    }
}
```

Výpis 7.5: XML parsovanie procesu

Takýmto spôsobom získame všetky informácie o artefakte, ktoré sú dôležité na porovnávanie rozdielov.

7.2.1 Grafické komponenty

Pokiaľ ide o element typu process a BPD, všetky informácie si ukladáme do grafu. Graf je objekt v ktorom uchovávame údaje o názve procesu, jeho parametroch (vstupných, výstupných, privátnych), list objektu Lane, ktorý obsahuje swimlanes procesu a objekt Model, v ktorom sú uložené všetky aktivity a eventy ako vrcholy grafu a hrany ako komunikačné spojovníky medzi aktivitami, bránami a udalosťami.

Pokiaľ sa jedná o BPD artefakt, aktivity a udalosti sú označené elementom <flowObject>, ktoré sa nachádzajú v elemente <lane>. Z tohto elementu získame informácie o pozícii, ktorá je definovaná na ose x a y. Tieto hodnoty si uložíme do Java objektu *Point(int x, int y)*. Ďalej získame identifikátor, názov a typ komponenty. Jednotlivé typy komponent sú pomenované v enumerated triede Type, takže každému typu pridelíme hodnotu z tejto triedy. Tieto informácie tvoria konštruktor jednotlivých vrcholov. Vrchol je definovaný abstraktnou triedou, ktorá má ďalšie atribúty ako vstupné a výstupne parametre, pripojené servisné, integračné služby a procesy, pre a post priradenia. Túto triedu potom dedia vrcholy, ktoré vytvárajú konkrétnu komponentu typu brána, aktivita a udalosť. Po parsovaní relevantných údajov pre danú komponentu, tieto hodnoty nastavíme vrcholu, ktorý sme vytvorili.

V prípade elementu typu process, komponenty sú uložené v elemente <item> z ktorého rovnakým spôsobom získame informácie a uložíme do vrcholov. Tieto artefakty nemajú narozdiel od BPD procesov swimlanes.

Komunikačné spojovníky sú označnené ako <inputPort> a <outputPort>, ktoré znova obsahujú referenciu na identifikátor komponenty. Spojovníky ukladáme do objektu Edge, kde konštruktor tvorí identifikátor zdrojového vrcholu a identifikátor cieľového vrcholu.

Po získaní všetkých informácií ukladáme vytvorený graf do dvoch HashMáp s kľúčom identifikátoru artefaktu a hodnotou graf podľa identifikátoru, či sa jedná o súbory staršej alebo novšej verzie.

7.2.2 Textové komponenty

Textové komponenty môžeme chápať ako artefakty aplikácie ktoré niesú nijako graficky zobrazené. Príkladom je UCA, business objekt a účastníci procesu. Práca s týmito komponentami je jednoduchšia, pretože stačí získať potrebné informácie na porovnávanie, ktoré uložíme do poľa Stringov. Nakoniec si uložíme hodnoty do HashMapy s kľúčom identifikátora artefaktu a hodnotou poľa Stringov.

7.3 Porovnávanie

7.3.1 Porovnávanie grafov

Po parsovaní všetkých artefaktov, v ktorých nastala zmena inicializujeme triedu Compare, ktorá uložené hodnoty porovnáva. Metoda na porovnávanie

grafov berie na vstup dva argumenty - HashMapu grafov starej verzie a HashMapu grafov novej verzie. Metoda najprv iteruje nad HashMapou staršej verzie, kde podľa podmienky či sa nachádza daný kľúč v HashMape novšej verzie získa hodnotu - teda graf HashMapy staršej verzie a graf novšej verzie. Nasledujúci výpis popisuje porovnávanie grafov:

```
public void compareGraphs(HashMap oldGraph, HashMap newGraph) {
    Iterator it = oldGraph.entrySet().iterator();
    while (it.hasNext()) {
        HashMap.Entry pair = (HashMap.Entry) it.next();
        if (newGraph.containsKey(pair.getKey())) {
            Graph g1 = (Graph) pair.getValue();
            Graph g2 = (Graph) newGraph.get(pair.getKey());
            Model m1 = g1.getModel();
            Model m2 = g2.getModel();
            compareVertices(m1.getAddedCells(),
                m2.getAddedCells());
            compareEdges(m1.getAddedEdges(), m2.getAddedEdges());
            if (!g1.getName().equals(g2.getName())) {
                g2.setNameChanged(true);
            }
        }
    }
}
```

Výpis 7.6: Porovnávanie grafov

Na porovnávanie vrcholov sa volá metoda compareVertices, ktorá berie argumenty typu list vrcholov staršej verzie a list vrcholov novšej verzie. Táto metoda prechádza znova najprv všetky vrcholy staršej verzie a hľadá, či sa daný vrchol nachádza aj v novšej verzii. Pokiaľ áno, porovnávajú sa hodnoty vrcholu (parametre, služby atp.).

Okrem aktivity typu script, všetky atribúty sa porovnávajú podľa toho, či sa rovnajú hodnotou. Script obsahuje zdrojový kód, ktorý môže byť často dlhý a zmeny v ňom niesú na prvý pohľad viditeľné. Na porovnávanie zdrojového kódu používame diff - založený ma Myerovom algoritme, opísaný v kapitole 4.5, ktorého výstupom je zmena označená symbolmi „++” pre pridanie a „--” pre vymazanie.

Ak nastala zmena v nejakom atribúte, vrchol sa označí boolean identifikátorom true ako zmenený. Naopak ak sa vrchol v novšej verzii nenachádza, znamená to, že bol zmazaný a označí sa podobne ako pri zmene identifikátorom za vymazaný. Na rovnakom princípe sa porovnávajú a označujú vrcholy v novšej verzii s tým rozdielom, že ak sa vrchol nenachádza v staršej verzii označí sa za pridaný.

Po porovnaní vrcholov sa zavolá metoda compareEdges na porovnanie hran, ktorá berie na vstup list hran staršej verzie a list hran novšej verzie. Rovnako ako pri ostatných porovnávaniach, najprv porovnávame hrany staršej verzie s hranami novšej verzie a pokiaľ sa daná hrana nenachádza v novej verzii, označí sa za vymazanú. Nápodobne porovnáme hrany novšej a staršej s tým, že ak sa tam nenachádza označí sa za pridanú.

Pretože sa môžu zmeniť aj názvy procesov nakoniec skontrolujeme, či sa rovnajú, ak nie, označia sa za zmenené.

7.3.2 Porovnávanie textu

Po dokončení porovnávania grafických komponent sa začnú porovnávať komponenty textové. Porovnávacia metoda znova berie na vstup HashMapu staršej verzie a HashMapu novšej verzie. Začneme iterovať nad HashMapou staršej verzie, kde za podmienky, že sa nachádza rovnaký kľúč aj v druhej HashMape, začneme tieto dva listy medzi sebou porovnávať. Pokiaľ sa nejaký String nenechádza aj v druhej verzii, označí sa ako „--" čiže sa považuje za zmazaný. Po iterácii všetkých prvkov sa tieto zmeny zapíšu do jedného Stringu. Ak sa kľúč nezhoduje s ani jedným z kľúčov druhej HashMapy, znamená to, že bol daný objekt vymazaný a tak ho aj označíme.

Nápodobne rovnakým spôsobom iterujeme nad HashMapou novšej verzie s rozdielom, že ak sa daný prvok nenachádza v staršej verzii, označí sa ako „++" za pridaný. Ak sa kľúče nezhodujú, objekt bol pridaný a označíme ho ako „New File".

7.4 Vykresľovanie

Posledným krokom je vykresľovanie grafických a textových komponent. Metoda na vykreslenie má ako vstupné argumenty HashMapu grafov staršej verzie a HashMapu grafov novšej verzie. Iterujeme nad HashMapami oboch verzií a v každom grafe prechádzame cyklusom všetky vrcholy a hrany, ktorým nastavíme vzhľad podľa typu komponentu a ich vlastností, ktoré získali pri porovnávaní.

Na vykresľovanie grafov, komponenty typu aktivita sú v tvare obdlžníku, udalosť v tvare kruhu a brána v tvare kosoštvorca. Hrany grafu sú vykreslené ako čiary. Swimlanes grafu sú znova obdlžníky. Komponenty poslúchajú na udalosť kliknutia myšou a vyvolávajú otvorenie dialógového okna, v ktorom sa nachádzajú informácie o danej komponente. Grafy sa vykresľujú pod seba, najprv graf staršej verzie a pod ním graf novšej verzie daného artefaktu.

Na vykreslenie textových komponent sa Stringy ukladajú do JavaFX objektu Text a rovnaké artefakty sa vykresľujú vedľa seba, pričom vedľa názvu artefaktu je identifikátor, či sa jedná novšiu alebo staršiu verziu.

7.5 Zhrnutie kapitoly

Výsledkom implementácie je spustiteľný program, ktorého výstupom je grafické rozhranie obsahujúce procesy a textové komponenty v ktorých nastala zmena.

Prvú časť implementácie tvorí dekomprimovanie súborov a následne uloženie do adresára. Program iteruje nad všetkými súbormi oboch verzií a ukladá si informáciu o ich veľkosti. Veľkosti rovnakých súborov sa potom navzájom porovnajú a pokiaľ sa veľkosť súboru zmenila, začneme dané súbory parsovať. Po získaní všetkých relevantných údajov, program začne porovnávať hodnoty

artefaktov a označí tie, ktoré sa zmenili. Nakoniec sa artefakty a ich zmeny vykreslia do užívateľského rozhrania.

Porovnávaný artefakt je zobrazený v podobe akú mal v staršej verzii a pod ním v podobe akú má v novšej verzii. Tento artefakt je vykreslený pod sebou a užívateľ vidí rozdiely aké nastali na komponentách, ktoré sú vykreslené farebne.

Kapitola 8
Testovanie

V tejto kapitole opisujem spôsob a postup testovania implementácie riešenia, popis testovacej aplikácie, testovacie scenáre a výsledky testov.

8.1 Vstupné podmienky

Testovacia procesná aplikácia je vytvorená v IBM Process Designeri, ktorý beží na servery umiestnenom v sieti ČVUT. Na spustenie nástroja pre porovnávanie je nutné mať nainštalovanú Javu JRE (Java Runtime Environment).

8.2 Testovacia aplikácia

Testovacia procesná aplikácia bola vytvorená za cieľom otestovať zdrojový kód, ktorý má porovnávať dve verzie aplikácie. Procesnú aplikáciu som vytvorila v IBM Process Designeri s názvom TST:TESTAPP (TSTAPP). V tejto aplikácii som vytvorila jednoduchý proces na prijímanie zamestnancov na novú pracovnú pozíciu.

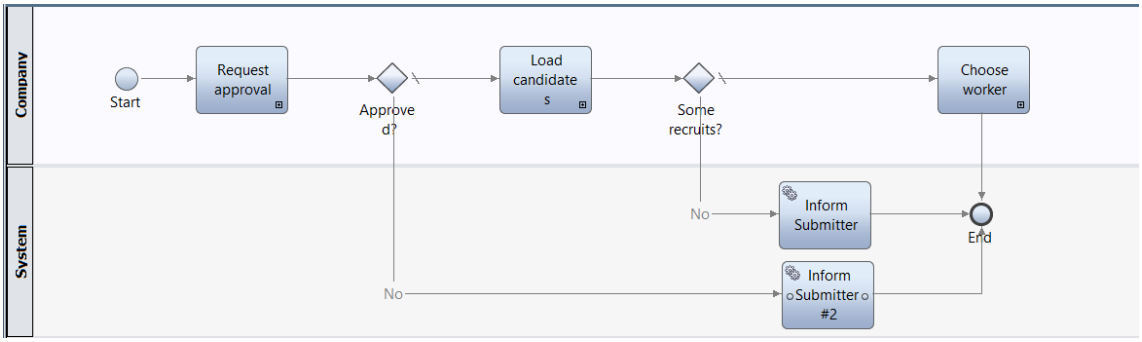
Proces obsahuje čo najviac artefaktov, ktoré je možné použiť pri vytváraní procesnej aplikácie. Proces tvoria dve BPD (hlavné) procesy, ktoré sa skladajú z niekoľkých podprocesov. Každá aktivita procesu je napojená na nejakú službu - General System Service, Integration Service, Human Service. Účastníci procesu sú napojený na swimlanes BPD. Aplikácia obsahuje dve Business objekty, ktoré sa používajú ako vstupné alebo výstupné parametre jednotlivých aktivít. Na vyvolanie spustenia určitého procesu som použila UCA. V aplikácii chceme porovnať zmenu v procesoch aj zmenu v Business objektoch a účastníkoch.

Očakávaným výstupom implementovaného nástroja je grafické zobrazenie zmien popísaných v nasledujúcich podkapitolách.

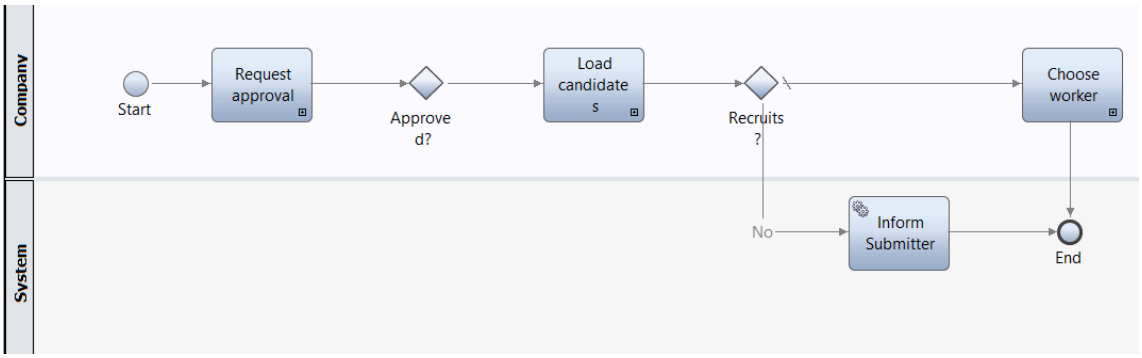
Testovacie scenáre vychádzajú z prípadov užitia.

8.2.1 TS1: Porovnávanie BPD

Scenár popisuje BPD proces, ktorý na obrázku 8.1 popisuje vzhľad procesu v staršej verzii a obrázok 8.2 zobrazuje proces v novšej verzii.



Obrázok 8.1: Hiring New Employee - BPD proces



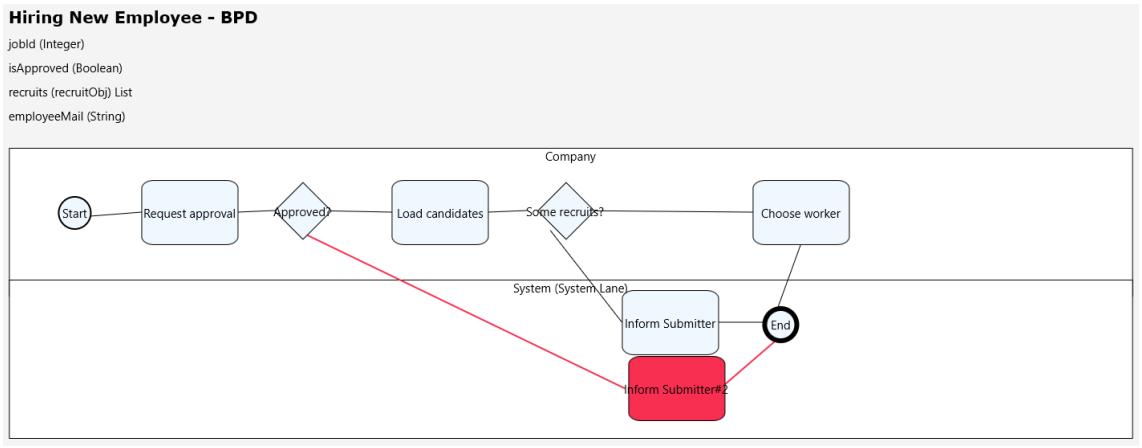
Obrázok 8.2: Hiring New Employee - BPD proces

8.2.2 Očakávaný výsledok

Očakávame, že systém zaznamená odstránenie aktivity „Inform Submiter2" a hran.

Skutočný výsledok

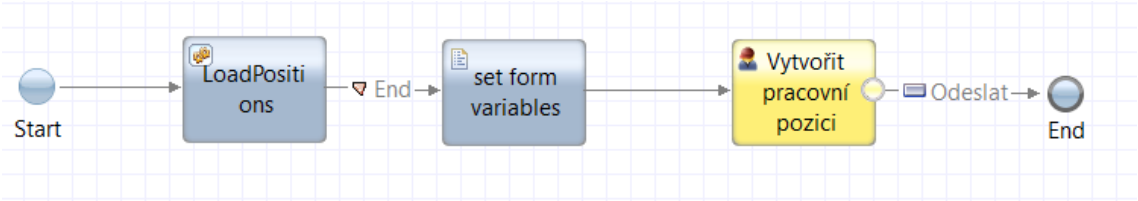
Aplikácia zmenu zaznamenala a vykreslila ju.



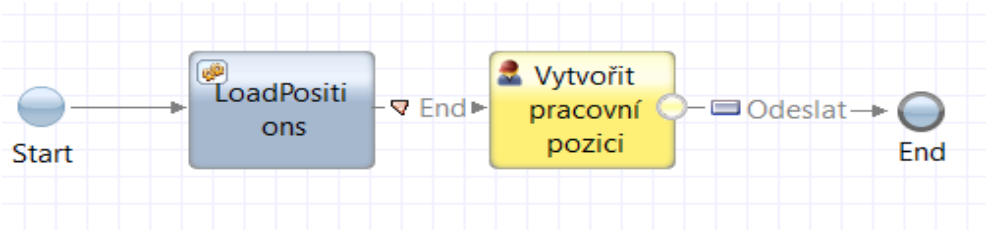
Obrázok 8.3: Hiring New Employee - zmena v starej verzii

8.2.3 TS2: Porovnávanie Human Services

Testovací scenár popisuje užívateľskú službu na vytváranie pracovnej pozície. Na obrázku 8.5 vidíme, že aktivita „set form variables" bola odstránená a okrem toho sa zmenil názov danej služby.



Obrázok 8.4: Add Job Requirement - Human Service



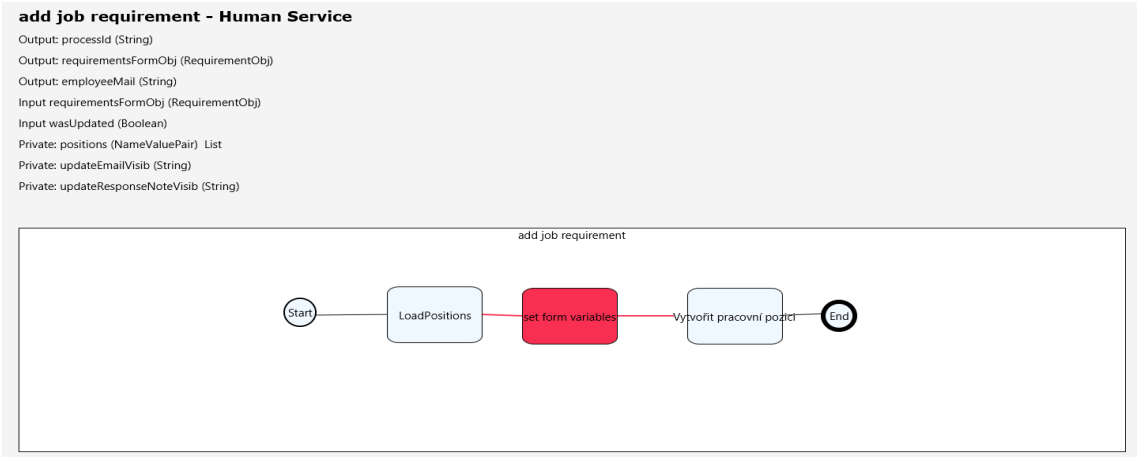
Obrázok 8.5: Add Job Requirement - Human Service

Očakávaný výsledok

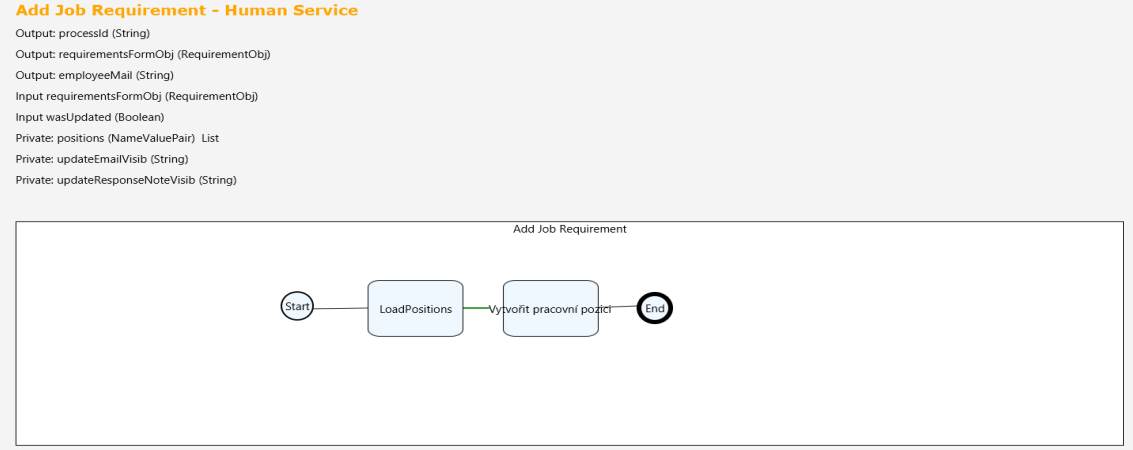
Očakávame, že aplikácia zaznamená odstránenie aktivity a zmenu v názve služby.

Skutočný výsledok

Systém naozaj zaznamenal odstránenie aktivity, ktorú zvýraznil červene na obrázku 8.6 a zmenu v názve na obrázku 8.7 vidíme oranžovo.



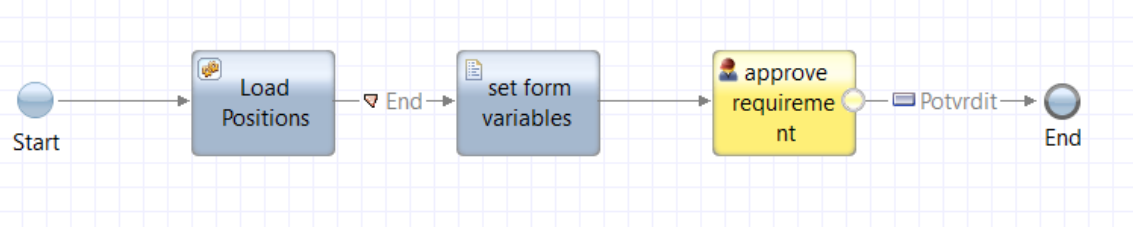
Obrázok 8.6: Add Job Requirement - zmena v starej verzii



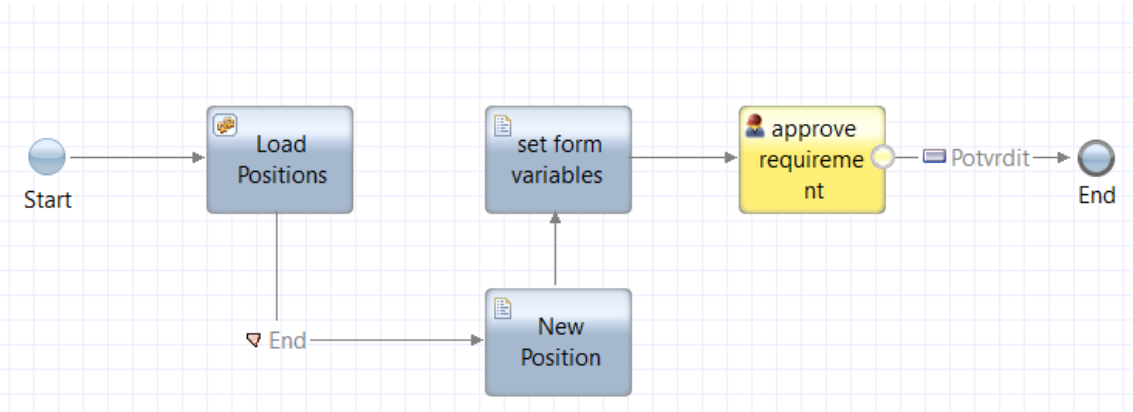
Obrázok 8.7: Add Job Requirement - zmena v novej verzii

8.2.4 TS3: Porovnávanie Human Services

Tento testovací scenár taktiež porovnáva uživateľskú službu na potvrdzovanie novej pracovnej pozície. V novšej verzii na obrázku 8.9 pribudla nová aktivita typu script a v aktivite „set form variables" bol zmenený zdrojový kód.



Obrázok 8.8: Approve Job Requirement - Human Service



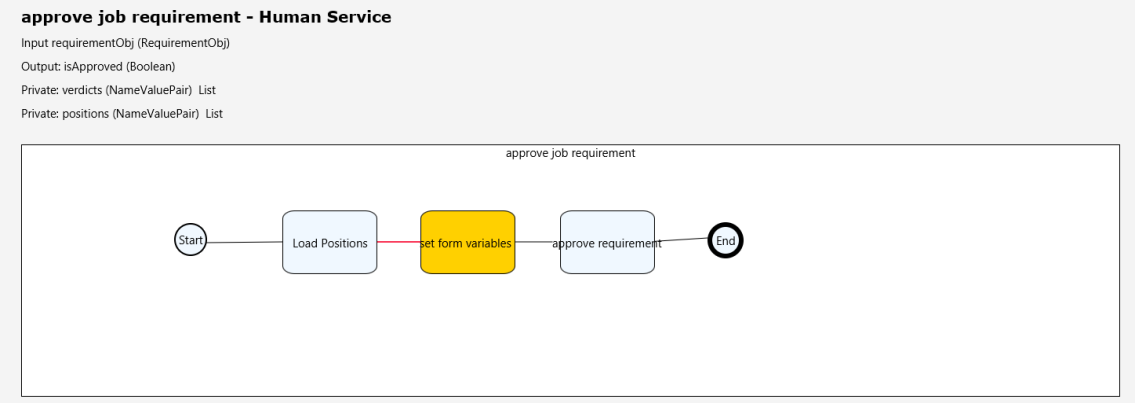
Obrázok 8.9: Approve Job Requirement - Human Service

Očakávaný výsledok

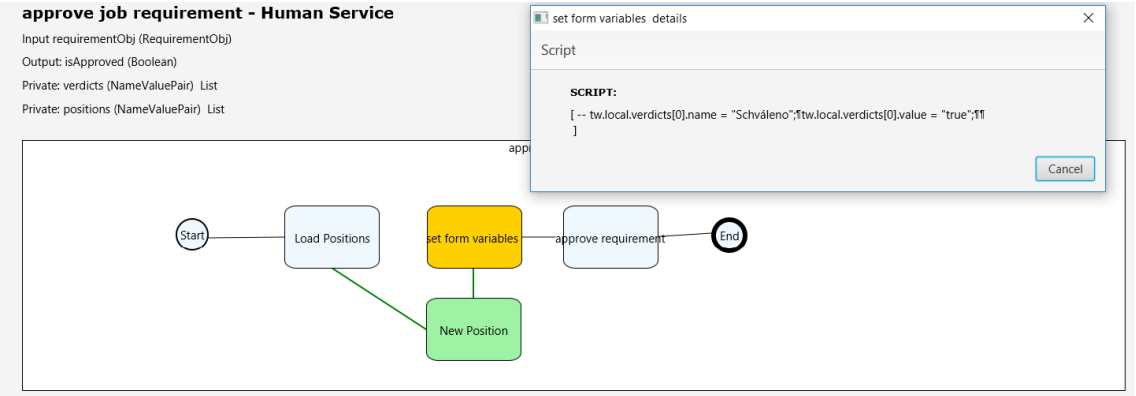
Očakávame, že systém zaznamená pridanie novej aktivity do procesu a zmenu v aktivite „set form variables".

Skutočný výsledok

Na obrázku 8.10 vidíme, že hrana je označená červene, takže bola vymazaná a aktivita „set form variables" je podľa očakávaní zvýraznená oranžovo, čo znamená, že v nej nastala zmena. Na obrázku 8.11 vidíme, že nová aktivita je zvýraznena zelene a po kliknutí na zmenenú aktivitu sa nám zmena v zdrojovm kóde zobrazila.



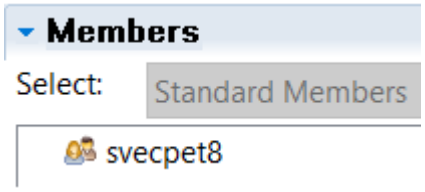
Obrázok 8.10: Approve Job Requierement - zmena v starej verzii



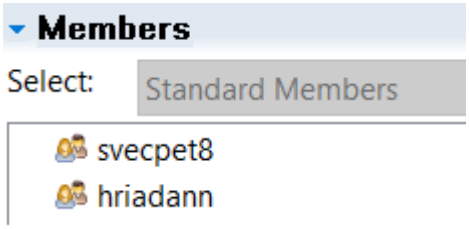
Obrázok 8.11: Approve Job Requierement - zmena v novej verzii

8.2.5 TS4: Porovnávanie Teamu

Tento testovací scénar opisuje Team, čiže skupinu účastníkov procesu. V staršej verzii na obrázku 8.12 vidíme, že aplikácia má len jedného účastníka. Na obrázku 8.13 pribudol nový účastník do skupiny štandardných členov.



Obrázok 8.12: Me - Participant



Obrázok 8.13: Me - Participant

Očakávaný výsledok

Očakávame, že aplikácia zaznamená nového člena v skupine a vypíše ho.

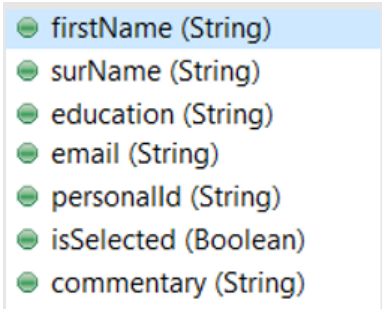
Skutočný výsledok

Na obrázku 8.14 vidíme, že účastník bol v novšej verzii zaznamenaný ako pridaný.

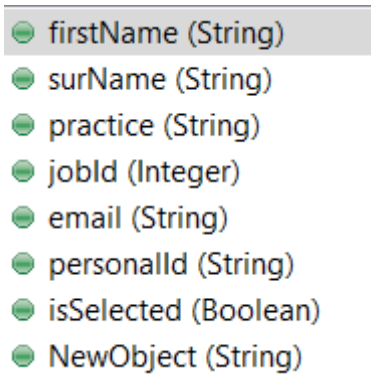
Obrázok 8.14: Participant: Me - zmena v novej verzii

8.2.6 TS5: Porovnávanie Business objektu

Posledný testovací scenár popisuje business objekt, v ktorom na obrázku 8.18 vidíme podobu staršej verzie a na obrázku 8.16 podobu novšej verzie.



Obrázok 8.15: recruitObj - Business Object



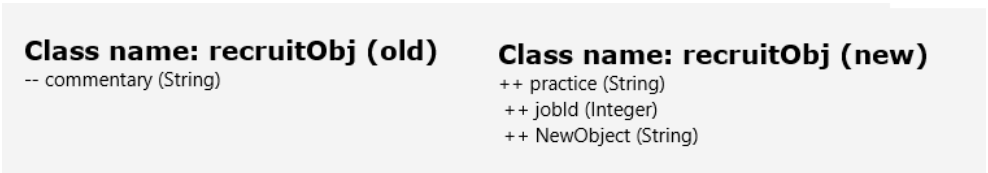
Obrázok 8.16: recruitObj - Business Object

Očakávaný výsledok

Očakávame, že aplikácia zaznamená odstránenie jedného atribútu v staršej verzii a pridanie 3 nových atribútov v novšej verzii.

Skutočný výsledok

Vidíme, že aplikácia zmeny zaznamenala na obrázku 8.17, kde v staršej verzii označil atribút „commentary" za odstránený a atribúty „practice", „jobId", „NewObject" v novšej verzii za pridané.



Obrázok 8.17: recruitObj - zmena v oboch verziách

8.3 Výsledky

Nasledujúca tabulka popisuje testovacie scenáre a ich úspešnosť.

Názov	Výsledok
TS1: Porovnávanie BPD	prešiel
TS2: Porovnávanie Human Service	prešiel
TS3: Porovnávanie Human Service	prešiel
TS4: Porovnávanie Teamov	prešiel
TS5: Porovnávanie Business objektu	prešiel

Tabuľka 8.1: Testovacie scenáre a ich úspešnosť

Na jednotlivých výstupoch vidíme, že aplikácia všetky zmeny vo verziach zaznamenala a vypísala. Na prvom riadku je vždy zvýraznený názov a typ artefaktu, ktorý je vykreslený. Pod názvom sa vypisujú parametre daného procesu a ich atribúty a pod nimi je samotný proces so zmenami.

8.4 Zhrnutie kapitoly

Na testovanie implementovaného nástroja som vytvorila procesnú aplikáciu v IBM Process Designeri na prijímanie zamestnancov na novú pracovnú pozíciu. Prvú verziu aplikácie som užila tak, že som vytvorila nový snapshot. Po niekoľkých úpravách som aplikáciu znova uložila a vytvorila ďalší snapshot. K úpravám aplikácie došlo takmer v každom artefakte, aby bolo možné čo najlepšie implemenetovaný nástroj otestovať. Jednotlivé snapshoty som z Process Centra exportovala ako twx súbor. Tento súbor je nutné prekonvertovať do ZIP archívu, aby ho program dokázal dekomprimovať.

Pri spustení programu som zadala ako vstupné argumenty cestu k súboru staršej verzie a cestu k súboru aktuálnej verzie. Po spustení aplikácia zobrazila grafické okno, v ktorom sú vykreslené všetky artefakty, ktoré som upravovala a zmeny v každom z nich sú zobrazené farebne aj symbolmi. Odstránené komponenty sú označené červene, pridané zelene a zmeny v komponente oranžovo. Zmeny v artefaktoch, ktoré sú popísané textovo, sa zobrazujú symbolom akcie a hodnotou. Po pridaní alebo odstránení celého artefaktu, aplikácia vypísala ich názvy a identifikátor toho, či bol artefakt zmazaný alebo naopak pridaný. Všetky testovacie scenáre boli vyhodnotené ako úspešné.

.

Kapitola 9
Záver

Cieľom práce bolo navrhnúť a implementovať prototyp nástroja na porovnávanie dvoch verzií procesnej aplikácie vytvorenej v IBM Process Designeri, ktorý by v budúcnosti mohol byť využívaný v komunite BPM vývojárov.

V teoretickej časti práce som opisovala existujúce riešenia systémov na porovnávanie verzií, vysvetľovala prečo je dôležité pri vývoji aplikácie prácu verzovať a vybrala jeden systém, ktorý som dôkladne analyzovala. Vybrané riešenie malo byť podkladom pre návrh bakalárskej práce. V ďalšej časti rešerše opisujem ako funguje verzovanie v IBM Business Process Manager. Najprv popisujem prostredie BPM, notáciu BPMN 2.0, ktorá slúži ako štandard pri vytváraní procesov a nakoniec píšem o tom, ako sa vytvárajú verzie v procesných aplikáciach a ako funguje porovnávanie medzi nimi.

V návrhovej časti som najprv opísala prípady užitia BPM vývojára a analyzovala, ako sú popísané jednotlivé artefakty v súboroch, ktoré vývojár získa pri exportovaní verzie. Každý artefakt je popísaný ako XML súbor, ktorý je nutné parsovať a tak získať dôležité údaje, ktoré porovnávame. Poznatky z rešeršovej časti o porovnávaní dvoch verzií pomocou funkcie diff som využila pri hľadaní rozdielov v komponentách so zdrojovým kódom. Najdôležitejším požiadavkom nástroja je vykresliť grafickú reprezentáciu zmien. Priebeh porovnávania som zachytila v diagrame aktivít a návrh modelu v diagrame tried.

Implementačná časť vychádza z prípadov užitia opísaných v návrhu riešenia a z modelu tried. V tejto časti opisujem aké sú potrebné vstupné podmienky do aplikácie a ako s nimi program narába. Veľkou časťou implementácie je samotné parsovanie zmenených súborov. Získané údaje sa ukladajú buď do grafu, pokiaľ sa jedná o artefakty popisujúce proces, alebo do textu, ak ide o artefakty popisaujúce objekty, učastníkov alebo UCA služby. Tieto informácie sa potom medzi sebou porovnávajú a každý rozdiel je zaznamenaný. Po porovnávaní údajov je nutné tieto údaje vykresliť do grafického rozhrania. Program iteruje nad porovnávanými dátami a postupne ich vykresľuje.

Toto implementované riešenie je nakoniec testované na vytvorenej procesnej aplikácii. V aplikácii som urobila viacero zmien aby som otestovala funkčnosť všetkých prípadov využitia.

Riešenie by mohlo byť využiteľné v komunite BPM vývojárov, ktorí by túto funkciu ocenili. Implementovaný nástroj je zatiaľ iba prototypom, pretože

pokryť požiadavky na porovnávanie zmien celej aplikikácie je nad rámec jednej bakalárskej práce. Stanovené ciele práce boli splnené.

Literatúra

[1] Atlassian Git Tutorial. Git diff [online], [cit. 2019-01-07]. Dostupné z: https://cs.atlassian.com/git/tutorials/saving-changes/git-diff.

[2] Eugene W. Myers. An o(nd) difference algorithm and its variations [online], 1986, [cit. 2019-01-04]. Dostupné z: http://xmailserver.org/diff2.pdf.

[3] IBM BPM. Architecture of ibm bpm [online], 2015, [cit. 2019-01-11]. Dostupné z: http://ibmbpmtechnical.blogspot.com/2015/12/architecture-of-ibm-bpm.html.

[4] IBM Knowledge Center. Building process applications [online], [cit.2019-01-06]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSFTBX_8.5.7/com.ibm.wbpm.wle.editor.doc/topics/proccessapp_highlevel_roadmap.html.

[5] IBM Knowledge Center. Managing and using toolkits [online], [cit. 2019-01-11]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSFTBX_7.5.1/com.ibm.wbpm.main.doc/managinglib/topic/managing_toolkits.html.

[6] Filip Stachecki. Bpmn 2.0 fundamentals and workshop [online], 2017, [cit. 2019-01-11]. Dostupné z: https://training-course-material.com/training/BPMN_2.0_Fundamentals_and_Workshop.

[7] Algorithms. Dynamic programming – longest common subsequence [online], 2016, [cit. 2019-01-08]. Dostupné z: https://algorithms.tutorialhorizon.com/dynamic-programming-longest-common-subsequence/.

[8] Livejournal alfedenzo. Patience diff, a brief summary [online], 2011, [cit. 2019-01-08]. Dostupné z: https://alfedenzo.livejournal.com/170301.html.

[9] Atlassian Git Tutorial. What is a version control [online], [cit. 2019-01-07]. Dostupné z: https://www.atlassian.com/git/tutorials/what-is-version-control.

[10] Open Hub. Compare repositories - open hub [online], 2018, [cit. 2019-01-04]. Dostupné z: https://www.openhub.net/repositories/compare.html.

[11] Inc Treehouse Island. Why you should switch from subversion to git [online], 2018, [cit. 2019-01-07]. Dostupné z: https://blog.teamtreehouse.com/why-you-should-switch-from-subversion-to-git.

[12] Ravishankar Somasundaram. *Git: Version Control for Everyone*. Packt Publishing Ltd., 2013.

[13] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[14] JGit. Histogram diff [online], 2012, [cit. 2019-01-08]. Dostupné z: http://download.eclipse.org/jgit/docs/jgit-2.0.0.201206130900-r/apidocs/org/eclipse/jgit/diff/HistogramDiff.html.

[15] IBM Corporation. Scaling bpm adoption: From project to program with ibm business process manager [online], Marec, 2012, [cit. 2019-01-08]. Dostupné z: ftp://ftp.software.ibm.com/software/in/events/softwareuniverse/resources/Scaling_BPM_Adoption_From_Project_to_Program_with_IBM_Business_Process_Manager.pdf.

[16] Object Management Group. Business Process Model and Notation [online], [cit. 2019-01-08]. Dostupné z: http://www.bpmn.org/.

[17] IBM Knowledge Center. Versioning process applications [online], [cit. 2019-01-08]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSFTBX_8.5.5/com.ibm.wbpm.wle.editor.doc/topics/cversions_inbpm.html.

Dodatok A

Zoznam použitých skratiek

BPM Bussines Process Manager
VCS Version Control System
LCS Longest Common Subsequence
SES Shortest Edit Script
BPMN Business Process Modeling Notation
XML Extensible Markup Language
UCA Undercover Agent
DOM Document Object Model
GUI Graphic User Interface

Dodatok B

Slovník pojmov

Commit je príkaz v Git, ktorý zaznamenáva zmeny do repozitára.

Working tree je adresár (a jeho súbory a podadresáre) v súborovom systéme, ktorý je priradený k repozitári.

Coach je artefakt, v ktorom je možné tvoriť užívateľské rozhranie v IBM Process Designeri.

Parse znamená rozoberanie súboru popísaného značkami, v práci poslovenčené ako parsovanie.

Dodatok C

Obsah priloženého CD

- implementation.zip - archív so zdrojovým kódom a testovacími procesnými aplikáciami
- BPHriadelovaAnnaMaria.pdf - text práce vo formáte pdf
- BPHriadelovaAnnaMaria.zip - archív obsahujúci obrázky a zdrojové tex súbory

Dodatok D

Užívateľská príručka

Užívateľská príručka je určená vývojarom BPM ako návod na použitie nástroja pre porovnávanie zmien medzi dvoma verziami aplikácie.

Export súborov v Process Centre:

- Výber aplikácie v ktorej chce užívateľ porovnávať zmeny.
- Výber prvého snapshotu a export ako súbor s príponou twx.
- Výber druhého snapshotu a export ako súbor s príponou twx.

Vybrané súbory musí užívateľ dekomprimovať a archivovať do súboru ZIP.

Užívateľ si otvorí príkazový riadok a prejde do adresára dist v ktorom sa nachádza jar súbor. Do príkazovej riadky napíšeme:

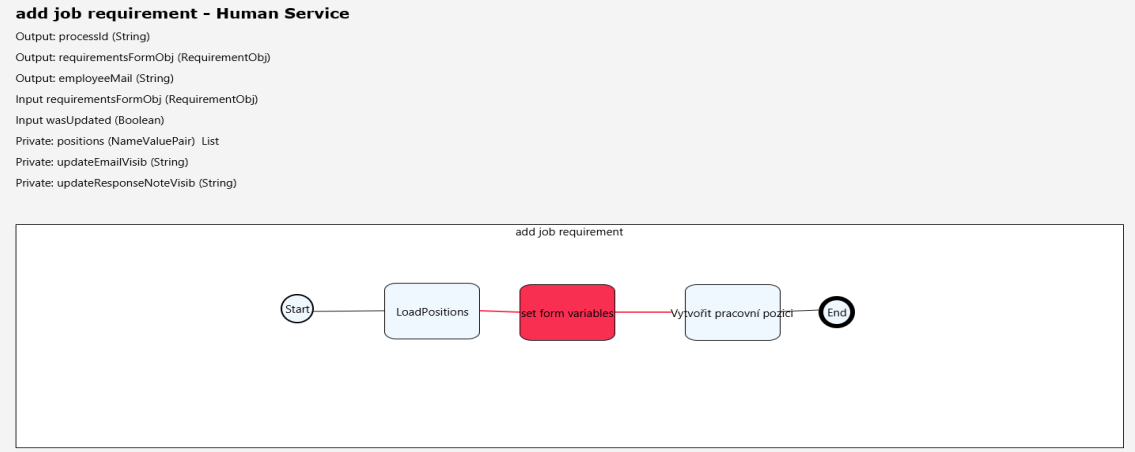
java -jar ”XMLparse.jar” <cesta> <cesta>

cestu k prvému súboru (staršia verzia) a cestu k druhému súboru (novšia verzia). Jar súbor zdrojový kód spustí so zadanými argumentami.

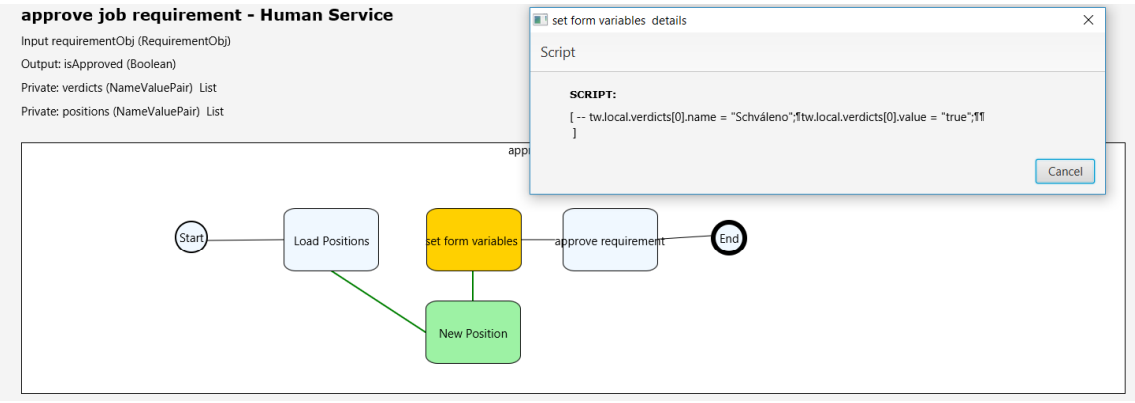
Po spustení sa užívateľovi zobrazí okno s grafickými aj textovými zmenami, ktoré vznikli medzi zadanými verziami. Vykreslené súbory sú vždy zoradené tak, že prvý súbor je starší a pod ním sa nachádza rovnaký súbor ale novší. Grafické zmeny sú popísané ako: prvý riadok je názov artefaktu a za pomlčkou jeho typ. Pod názvom sú vypísané premenné daného artefaktu, identifikované ako vstupné, výstupné alebo privátne, názov premennej, v zátvorke typ atribútu a identifikátor poľa. Pod premennými sa nachádza samotný proces danej verzie. Proces je vykreslený ako v IBM Process Designeri, kde užívateľ vidí rozdiely následovne:

- Červená farba identifikuje odstránenie aktivity alebo hrany v staršej verzii.
- Zelená farba identifikuje pridanie novej aktivity alebo hrany v novšej verzii.
- Oranžová farba identifikuje zmenu v aktivite.

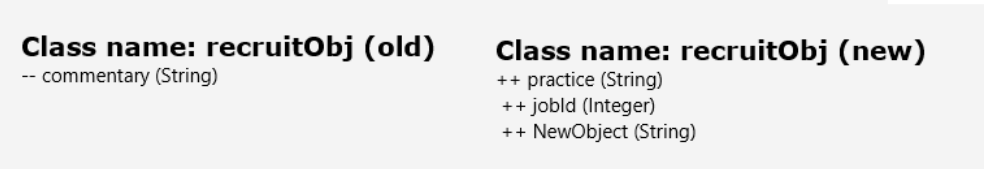
Na obrázkoch D.1 - D.3 sú zobrazené príklady rozdielov v grafickej podobe. Na každú aktivitu je možné kliknúť a po kliknutí sa otvorí užívateľovi dialógové okno s parametrami aktivity. Vstupné a výstupné premenné, pripojená služby alebo proces a typ aktivity. Zmeny v business objekte, UCA službe a týme sú popísané textovo a to tak, že staršia verzia je zobrazená naľavo a novšia napravo.



Obrázok D.1: Add Job Requirement - zmena v starej verzii



Obrázok D.2: Approve Job Requierement - zmena v novej verzii



Obrázok D.3: recruitObj - zmena v oboch verziách

Pokiaľ boli v staršej verzii nejaké súbory zmazané, program ho označí ako Deleted: <názov súboru>. Naopak pokiaľ v novšie verzii súbory pribudli, označia sa ako New: <názov súboru>.